SCIRun

Release 0.01

TBA

Sep 20, 2023

GETTING STARTED:

SCIRun is a problem solving environment developed by the NIH Center for Integrative Biomedical Computing at the University of Utah Scientific Computing and Imaging (SCI) Institute.

SCIRun 5 is a complete rewrite of the GUI front end and graphical components of SCIRun 4, including a more stable and efficient middle layer, with support for Python scripting.

Warning! SCIRun 5 is beta software, you may use for real science but beware of instability.

CHAPTER

ONE

BUILD

1.1 Platform Notes

At least C++11 64-bit compiler support is required.

1.1.1 Build requirements

OS X

- MacOS 11 or newer
- Apple clang 9.0.0 or newer
- Qt 5.15 or later
  - Download and run the desired Qt 5.x or 6.x installer. Make sure to turn off other versions and system configurations to save space and build time. Configure CMake for Qt 5. Optionally, install Qt through the package manager, brew.

    To install Qt 5, the command is:

    ```
    brew install qt@5
    ```

    To install Qt 6, the command is:

    ```
    brew install qt@6
    ```

    The installation directory for the CMake variable `Qt_PATH` will then be `/usr/local/Cellar/Qt/_TYPE_QT_VERSION_HERE_`.

Windows

- Tested on Windows 10
- Visual Studio 2017 or 2019
  - Using 2017, be sure to change the CMake platform to x64.
- Qt 5.15 or later
  - Download and run the Qt 5.15 installer. Make sure to turn off other versions and system configurations to save space and build time.

Linux

- Tested on Ubuntu 16.04 LTS, 18.04 LTS, 22.04 LTS, OpenSUSE Leap 42.1, Arch Linux
- gcc 7+
- Qt 5.15 or later
  - Download and run the desired Qt 5.x or 6.x installer. Make sure to turn off other versions and system configurations to save space and build time. Optionally, install Qt through your distro's package manager instead(apt on Ubuntu/Debian, pacman on Arch).

    To install Qt 5, the command on Ubuntu/Debian is:

    ```
    sudo apt-get install qt5-default
    ```

    To install Qt 6, the command on Ubuntu/Debian is:

    ```
    sudo apt-get install qt6-base-dev libqt6svg6-dev
    ```

    The installation directory for the CMake variable `Qt5_DIR`(for Qt5) or `Qt_PATH`(for Qt6) will then be `/usr/lib/x86_64-linux-gnu/cmake/`.

    To install Qt 5, the command on Arch is:

    ```
    sudo pacman -S qt5-base
    ```

    To install Qt 6, the command on Arch is:

    ```
    sudo pacman -S qt6-base
    ```

    The installation directory for the CMake variable `Qt5_DIR`(for Qt5) or `Qt_PATH`(for Qt6) will then be `/usr/lib/cmake/`.

All Platforms

- CMake (platform independent configuring system that is used for generating Makefiles, Visual Studio project files, or Xcode project files)
  - Tested with 3.4 and newer
  - Root cmake file is Superbuild/CMakeLists.txt.
  - Building in source directories is not permitted.
  - Make sure BUILD_SHARED_LIBS is on (default setting).

1.1.2 CMake Build Generators

- Windows
  - Visual Studio 2017 & 2019
- OS X
  - Unix Makefiles
  - Xcode
- Linux

- Unix Makefiles

1.2 Configuring CMake

Run CMake from your build (bin or other build directory of your choice) directory and give a path to the CMake Superbuild directory containing the master CMakeLists.txt file.

A bash build script (`build.sh`) is also available for Linux and Mac OS X to simplify the process. Usage information is available using the ***–help*** flag:

```
./build.sh --help
```

For example, on the command line if building from the default SCIRun bin directory:

```
cd bin
cmake ../Superbuild
```

The console version `ccmake`, or GUI version can also be used. You may be prompted to specify your location of the Qt installation. If you installed Qt in the default location, it should find Qt automatically.

1.2.1 Configuring SCIRun with Qt 5

Building SCIRun with Qt 5 requires additional input. Use the `Qt5_PATH` CMake variable to point to the Qt 5 build location. Look at the Qt install steps above for information about the directory. This can be done through the command line with a command similar to:

```
cmake -DQt5_PATH=path_to_Qt5_build/ ../Superbuild/
```

Or they can be set in the CMake GUI or with the `ccmake` function.

The command will be similar to the following:

```
cmake -DQt5_PATH=path_to_Qt5/5.15.1/clang_64/ ../Superbuild/
```

1.2.2 Configuring SCIRun with Qt 6

Building SCIRun with Qt 6 requires additional input. Set the Cmake variable `SCIRUN_QT_MIN_VERSION` to 6.X.X, where the version is less than the installed Qt 6 version. Use the `Qt_PATH` CMake variable to point to the Qt 6 build location. Look at the Qt install steps above for information about the directory. This can be done through the command line with a command similar to:

```
cmake -DQt_PATH=path_to_Qt6_build/ ../Superbuild/
```

Or they can be set in the CMake GUI or with the `ccmake` function.

The command will be similar to the following:

```
cmake -DQt_PATH=path_to_Qt6/6.4.2/clang_64/ -DSCIRUN_QT_MIN_VERSION="6.3.1" ../
↪Superbuild/
```

1.2.3 Configuring SCIRun with OSPRay

To use the OsprayViewer module, SCIRun needs to download and install Ospray during the build process, which is off by default. This is enabled with the `WITH_OSPRAY` flag. In the command line, it would look like:

```
cmake -DWITH_OSPRAY=True ../Superbuild/
```

1.3 Building SCIRun

After configuration is done, generate the make files or project files for your favorite development environment and build.

From the same command line, you can build with:

```
make
```

Append `-jN`, where N is the number of threads, to build multi-threaded. Note: A common problem during the first build is the Python build fails with multi-threading. If this happens, rebuild single-threaded.

Following the previous example, the SCIRun application will be built in `bin/SCIRun`.

1.4 Partial Rebuild

After SCIRun has finished building externals from the `bin/` folder, you can recompile just the internal SCIRun code by building in the `bin/SCIRun` folder. In the command line, it would look like:

```
cd bin/SCIRun
make
```

Append `-jN`, where N is the number of threads, to build multi-threaded.

1.5 Tagging Releases

On an OSX system, run script `release.sh` in the `src` directory with the release name in format ***beta.XX*** as a parameter.

CHAPTER

TWO

BASIC TUTORIAL

2.1 SCIRun Overview

This tutorial demonstrates how to build a simple SCIRun dataflow network.

2.1.1 Software requirements

SCIRun

All available downloads for SCIRun version and the SCIRunData archive are available from SCI software portal. Make sure to update to the most up-to-date release available, which will include the latest bug fixes.

Currently, the easiest way to get started with SCIRun version is to download and install a binary version for Mac OS X. Sources are also available for Linux, however this option is recommended only for advanced Linux users.

Unpack the SCIRunData archive in a convenient location. Recall from the User Guide that the path to data can be set using the environment variable or by setting in the *.scirunrc* file.

2.2 Simple Dataflow Network

2.2.1 Slice Field

The purpose of this section is to read, manipulate, and visualize a structured mesh dataset originating from SCIRunData.

Read Data File

Create a **ReadField** module by using the **Module Selector** on the left hand side of the screen. Navigate to **DataIO** subsection using the scroll bar in the Module Selector and instantiate a ReadField (Fig. 2.1). Recall from the **User Guide** that a module can also be selected by giving a text input into the filter in the Module Selector (Fig. 2.2).

Within the ReadField **user interface (UI)**, click the open button to navigate to the SCIRunData directory and select the dataset *volume/engine.nhdr* (Fig. 2.3). Notice that many different file formats can be imported by changing the file type within the ReadField selector window. When using Mac OSX El Capitan, press the options button in the ReadField selector window to change the file type. Change the file type to Nrrd file. The ReadField UI can be closed after selection to provide for a larger network viewing frame.

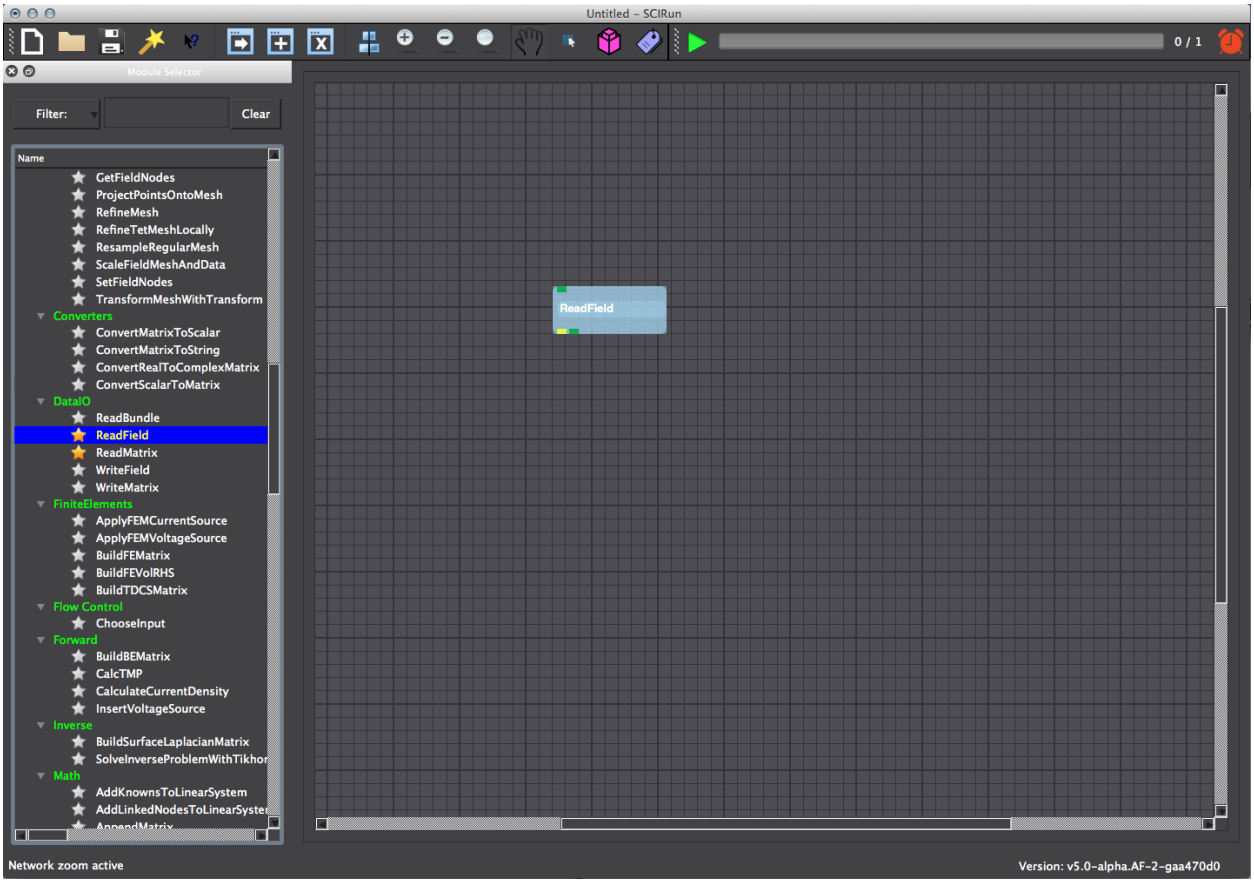


Fig. 2.1: Locate ReadField module using scroll bar in the Module Selector.

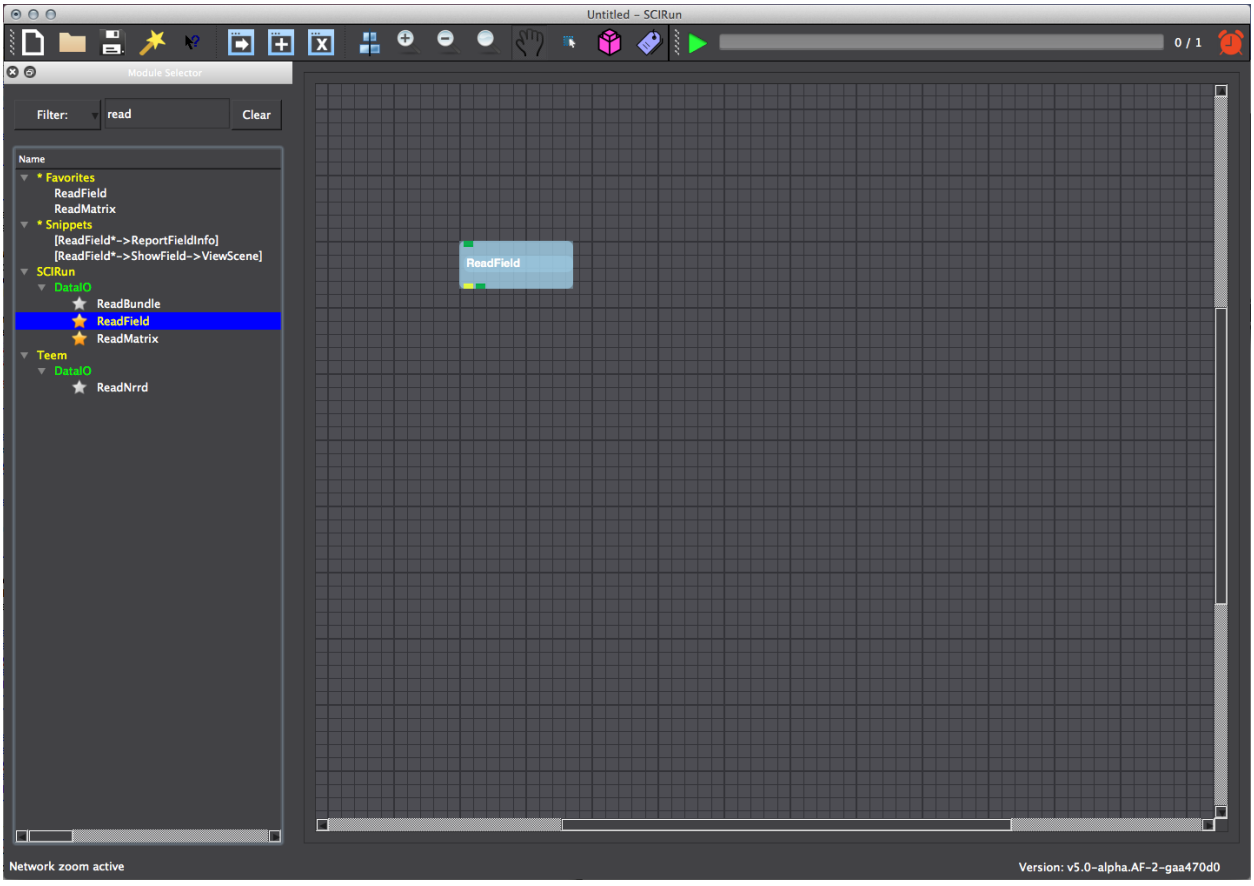


Fig. 2.2: Locate ReadField module using text input into filter.

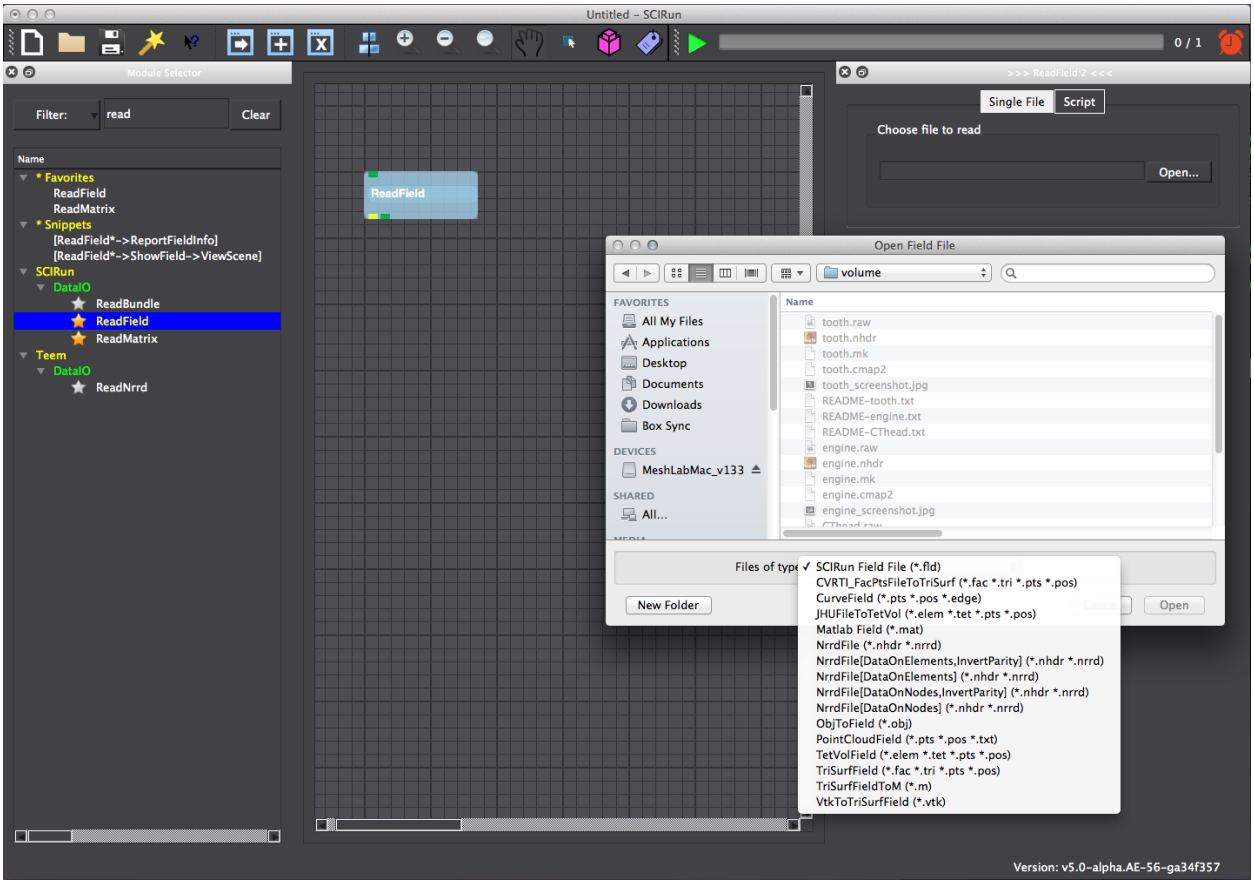


Fig. 2.3: The ReadField selector window can be used to select and read many data files.

Slice Field

Slice the engine field by node index along a given axis by instantiating the module **GetSlicesFromStructuredField-ByIndices** in the **NewField** category and connecting it to ReadField (Fig. 2.4). This can be done by using the Module Selector filter or scrolling through the list of modules in the Module Selector.

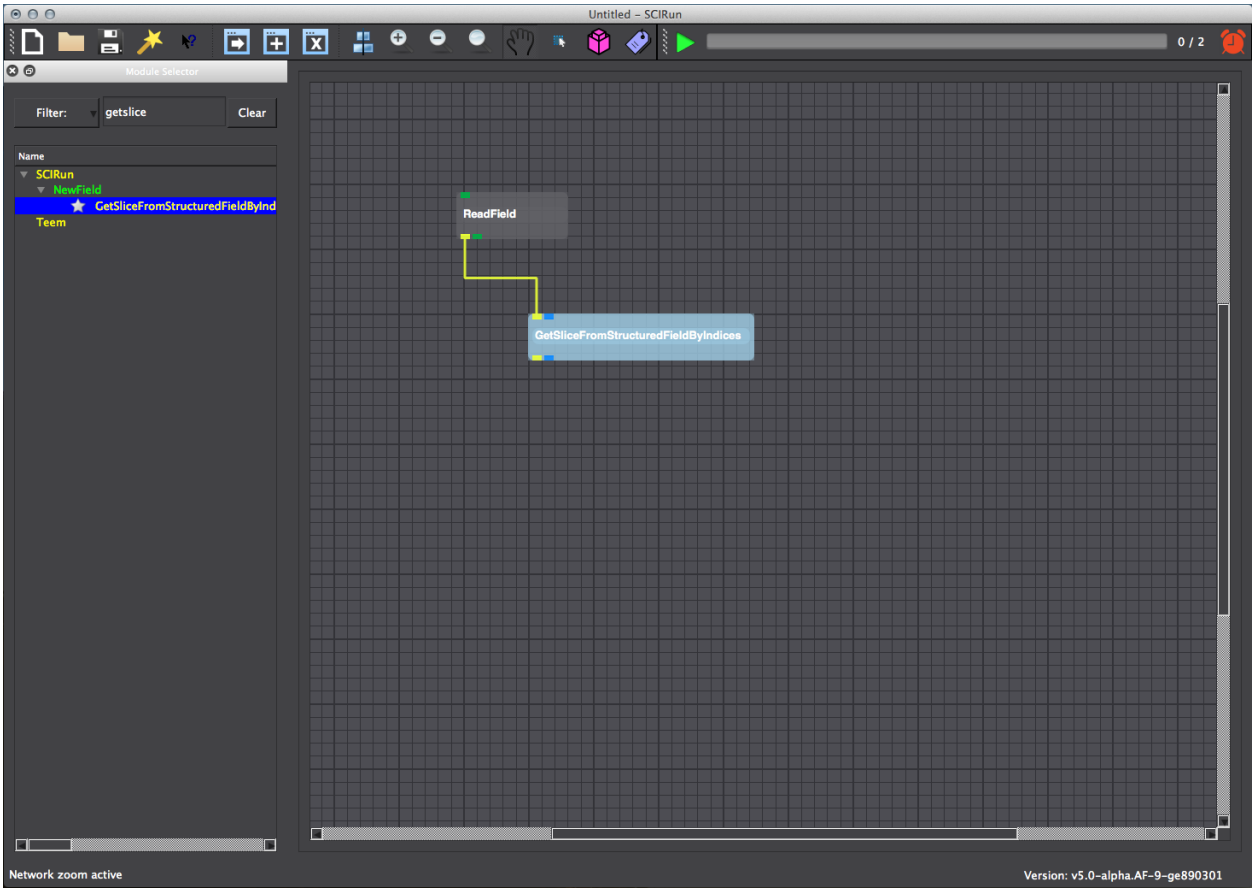


Fig. 2.4: Using the ReadField port's pop-up module menu to instantiate GetSliceFromStructuredFieldByIndices.

Visualize Field

To visualize the field geometry, instantiate module **ShowField** in the **Visualization** category and module **ViewScene** in the **Render** category (Fig. 2.5). ShowField takes a field as input, and outputs scene-graph geometry. ViewScene displays the geometry and allows a user to interact with the scene.

Apply a colored scale to the data values on the geometry using **CreateStandardColorMaps** and **RescaleColorMaps** modules in **Visualization** (Fig. 2.6). Colors can be manipulated using the CreateStandardColorMap UI and Rescale-ColorMap UI (Fig. 2.7). Change the coloring scheme to Blackbody using the drop-down menu in the CreateStandard-ColorMap UI.

Return to the default color scale. Use the sliders in the GetSlicesFromStructuredFieldByIndices UI to change slice position within the geometry. Compare with Fig. 2.6.

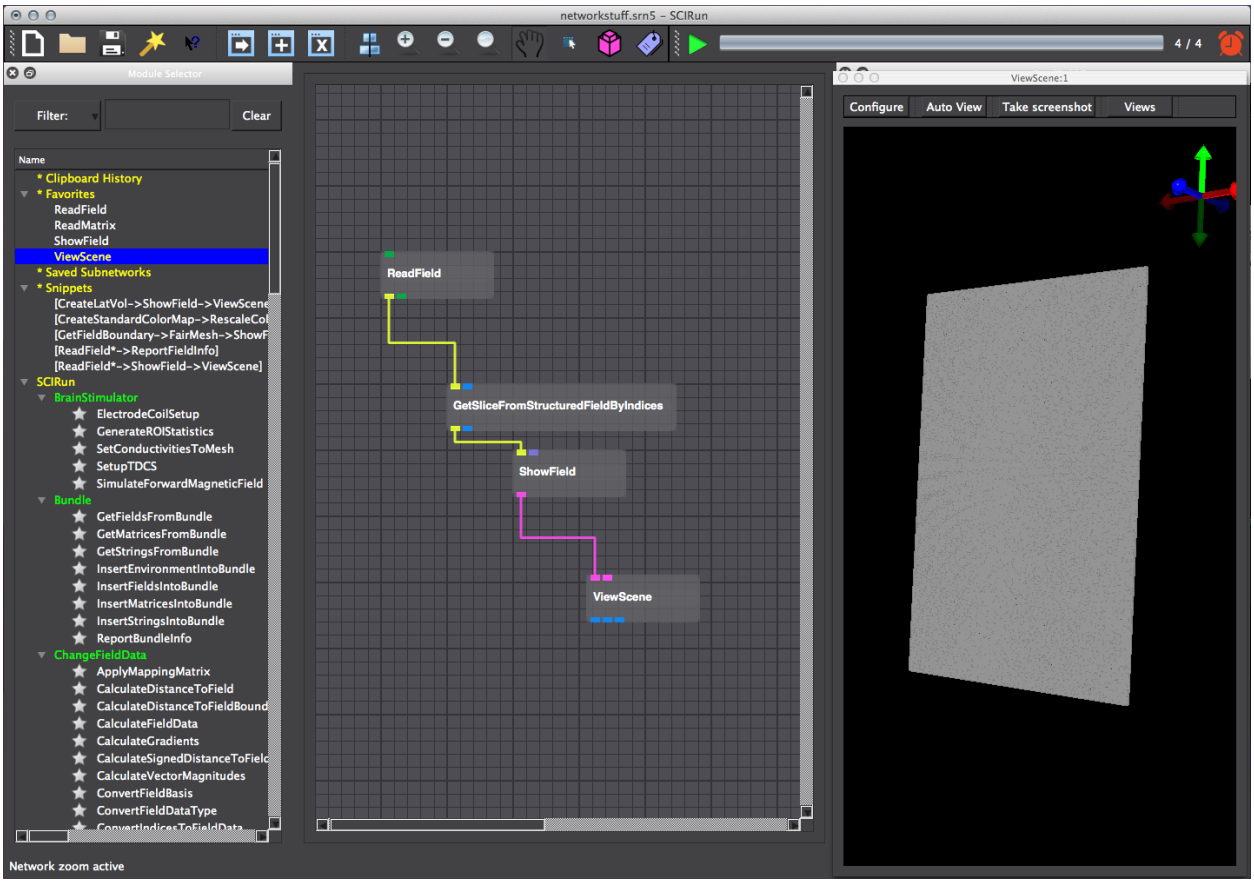


Fig. 2.5: SCIRun can be used to visualize the structured mesh.

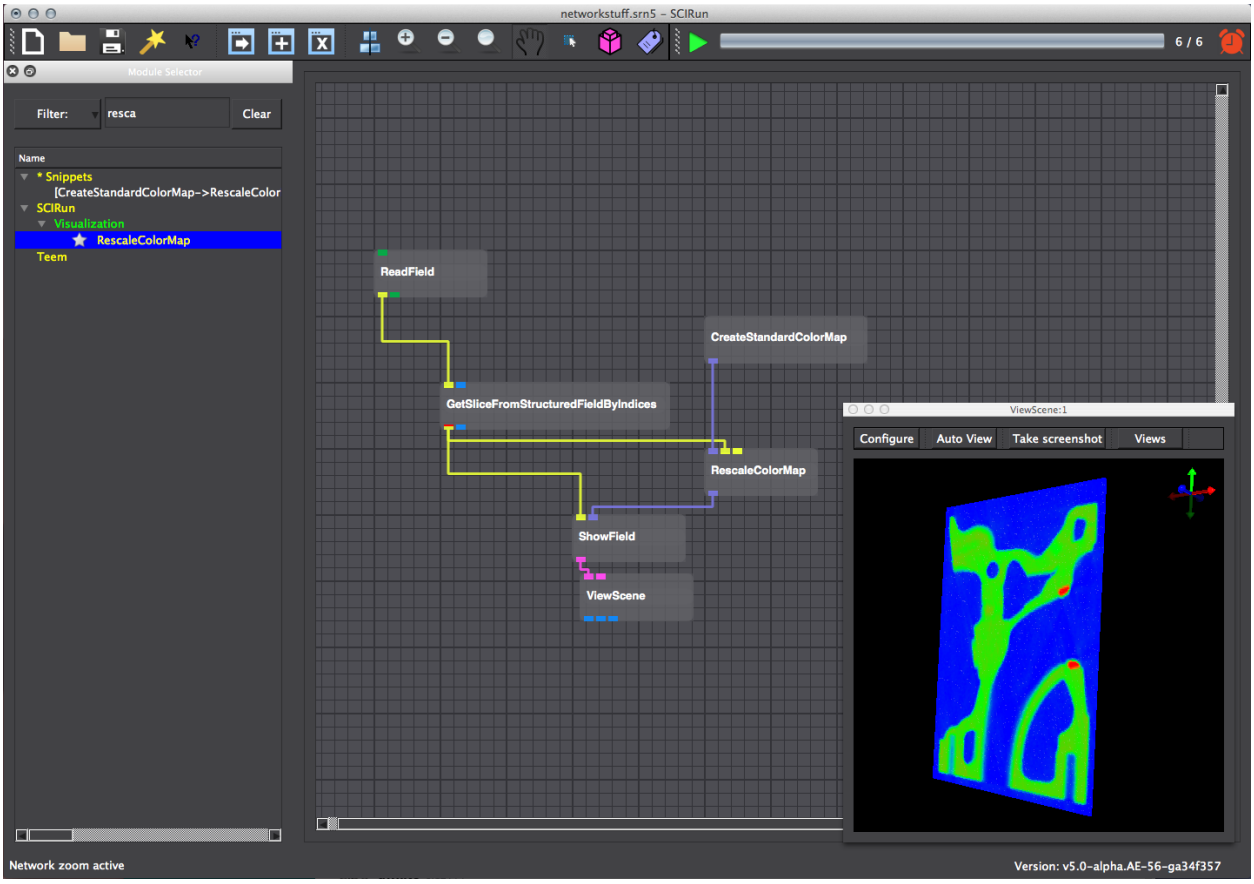


Fig. 2.6: Apply and rescale a colormap to data values on the geometry.

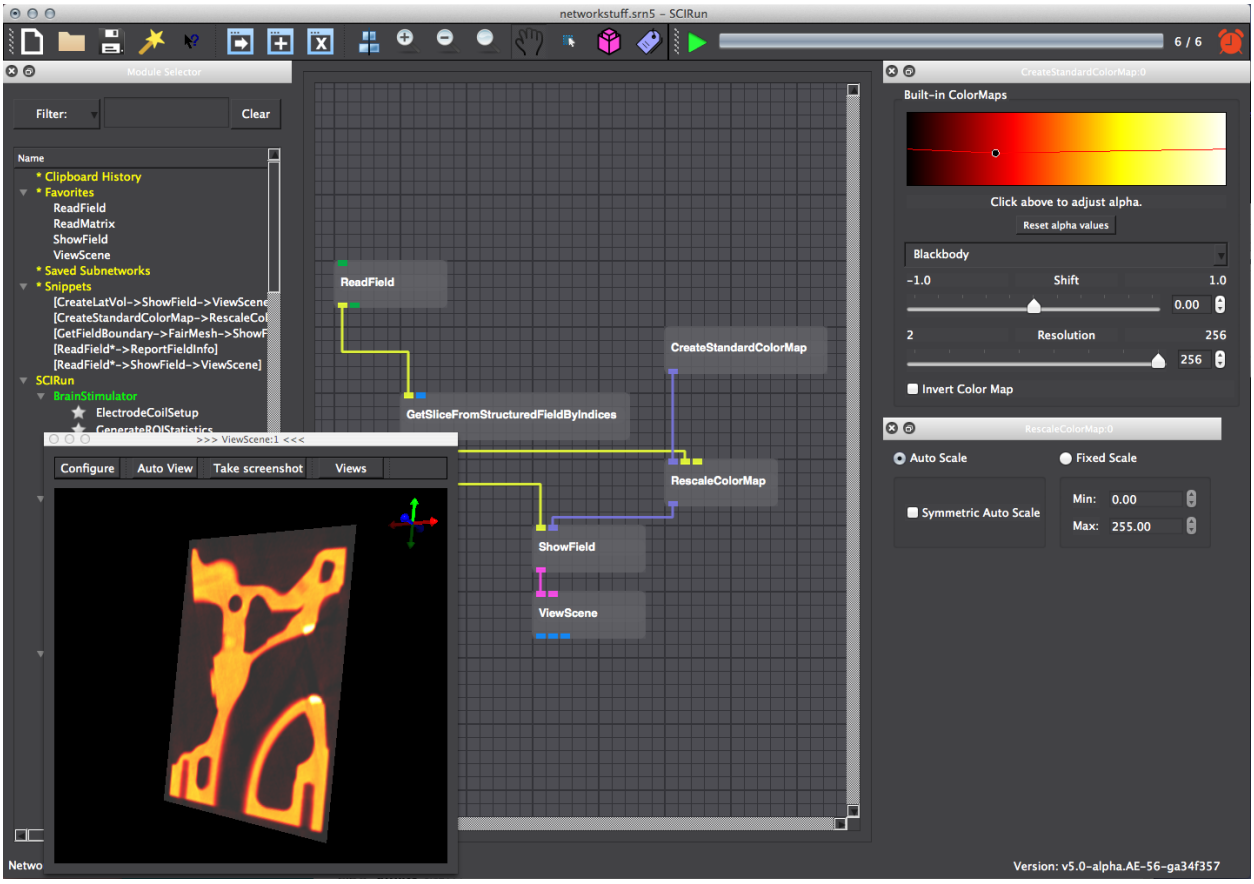


Fig. 2.7: Manipulate the color scaling using both the CreateStandardColorMaps and RescaleColorMaps modules.

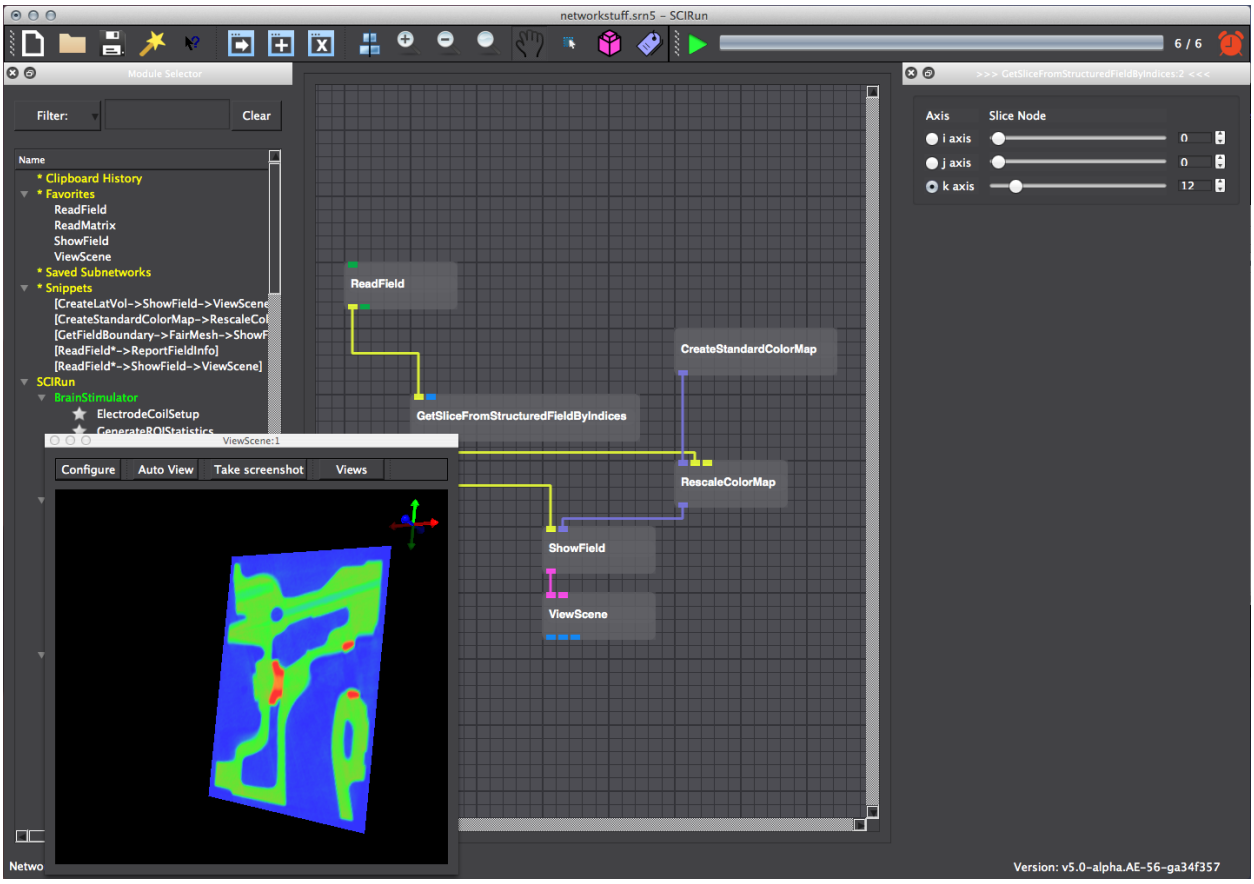


Fig. 2.8: Different cross sections can be visualized within the geometry using GetSlicesFromStructuredFieldbyIndices.

2.2.2 Show Bounding Box

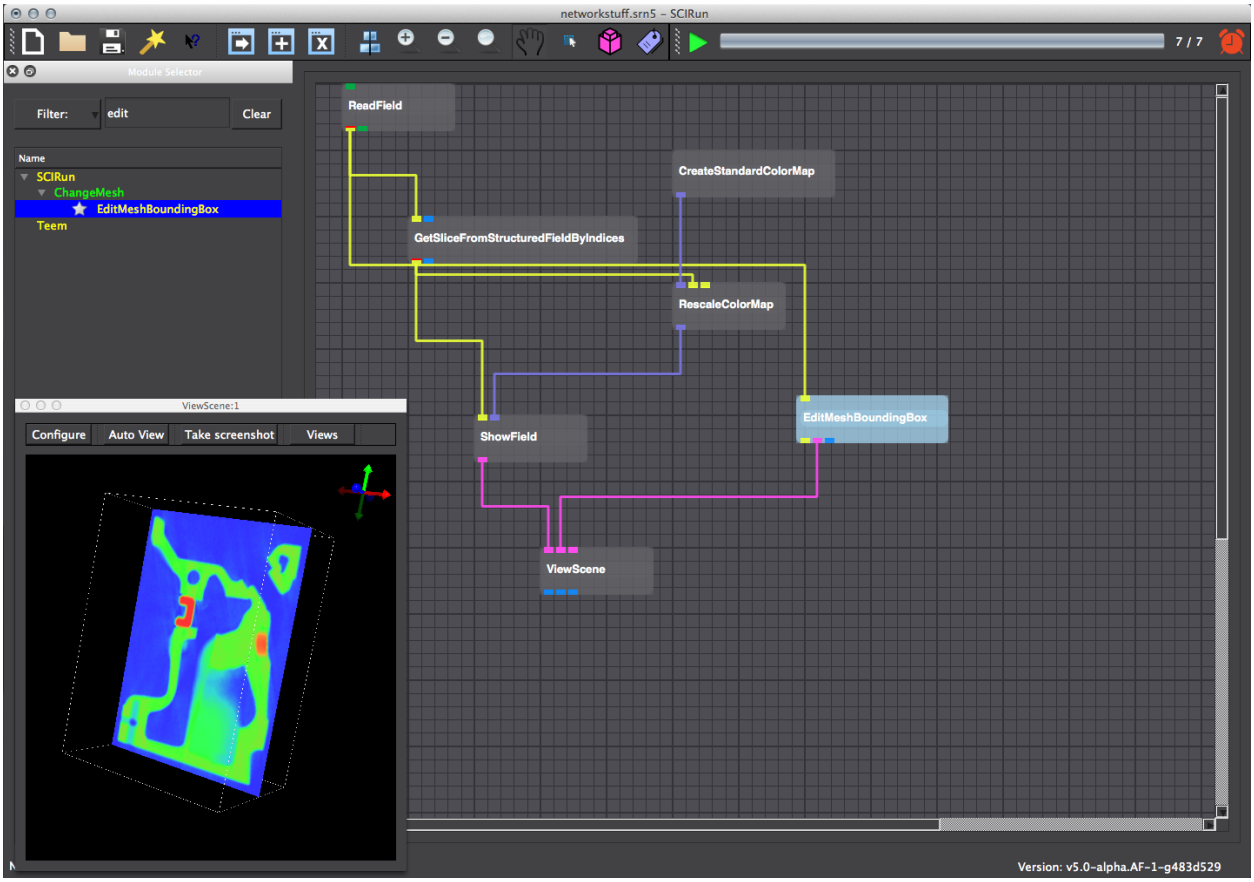


Fig. 2.9: Visualize the mesh's bounding box.

Add the **EditMeshBoundingBox** module under **ChangeMesh** (Fig. 2.9). Connect it to the ReadField module and direct the output to the ViewScene module. Execute the network to visualize the bounding box of engine.nhrd. Adjust the size of the bounding box by pressing the + or - buttons under Widget Scale in the EditMeshBoundingBox UI (Fig. 2.10).

2.2.3 Isosurface

Construct an isosurface from the field by instantiating and connecting a **ExtractSimpleIsosurface** module to the ReadField module. The isovalue must be changed within the ExtractSimpleIsosurface UI. Open the field information by clicking on the connection between the ReadField and ExtractSimpleIsosurface and press I to bring up information. Enter a value from within the data range like 120. Visualize the isosurface by connecting it to a new ShowField module ported into the ViewScene module (Fig. 2.11). Execute the network. Color isosurface output geometry by connecting the RescaleColorMap module to the ShowField module (Fig. 2.12). To better view the geometry, turn off the edges within the ShowField UI (Fig. 2.13).

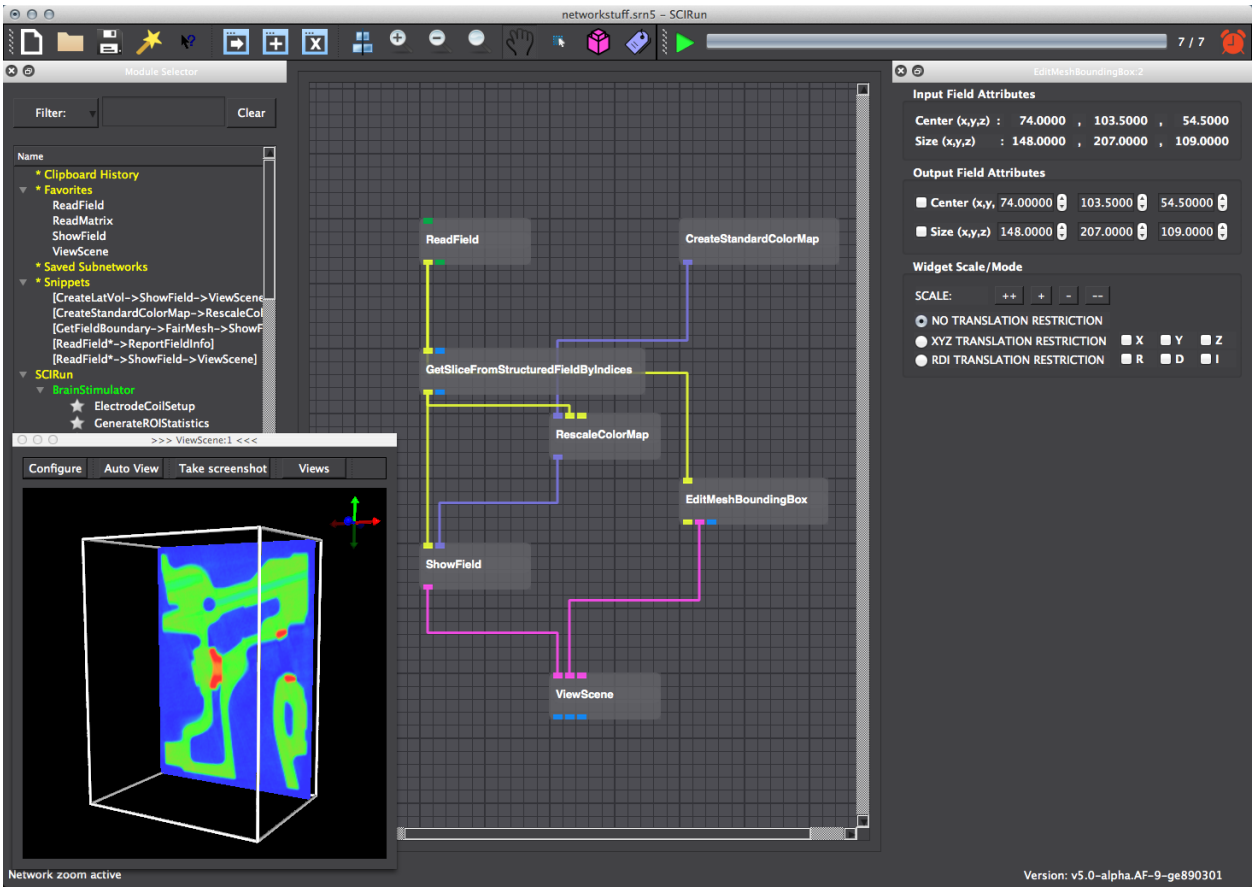


Fig. 2.10: Change the scale of the mesh's bounding box using the Scale Widget in the EditMeshBoundingBox UI.

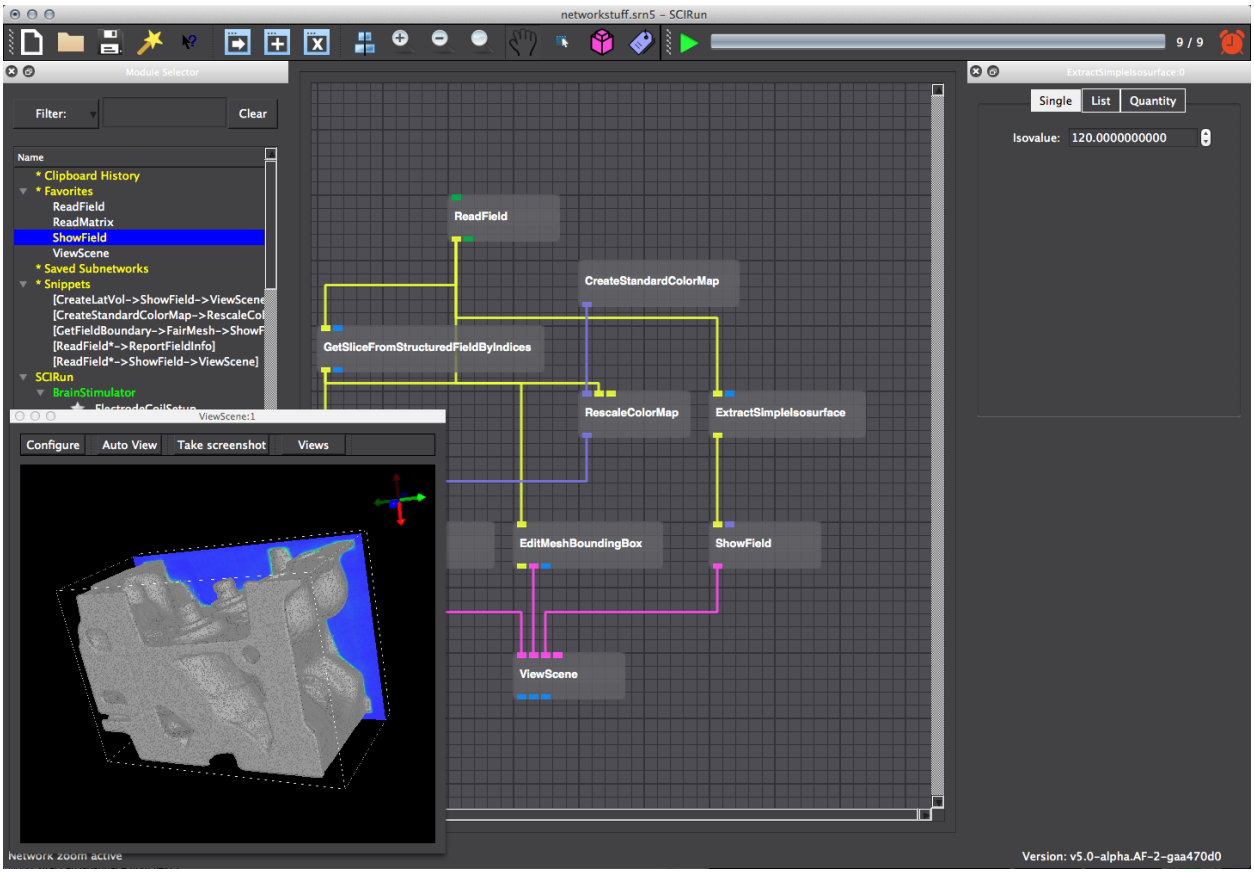


Fig. 2.11: Extract an isosurface from field.

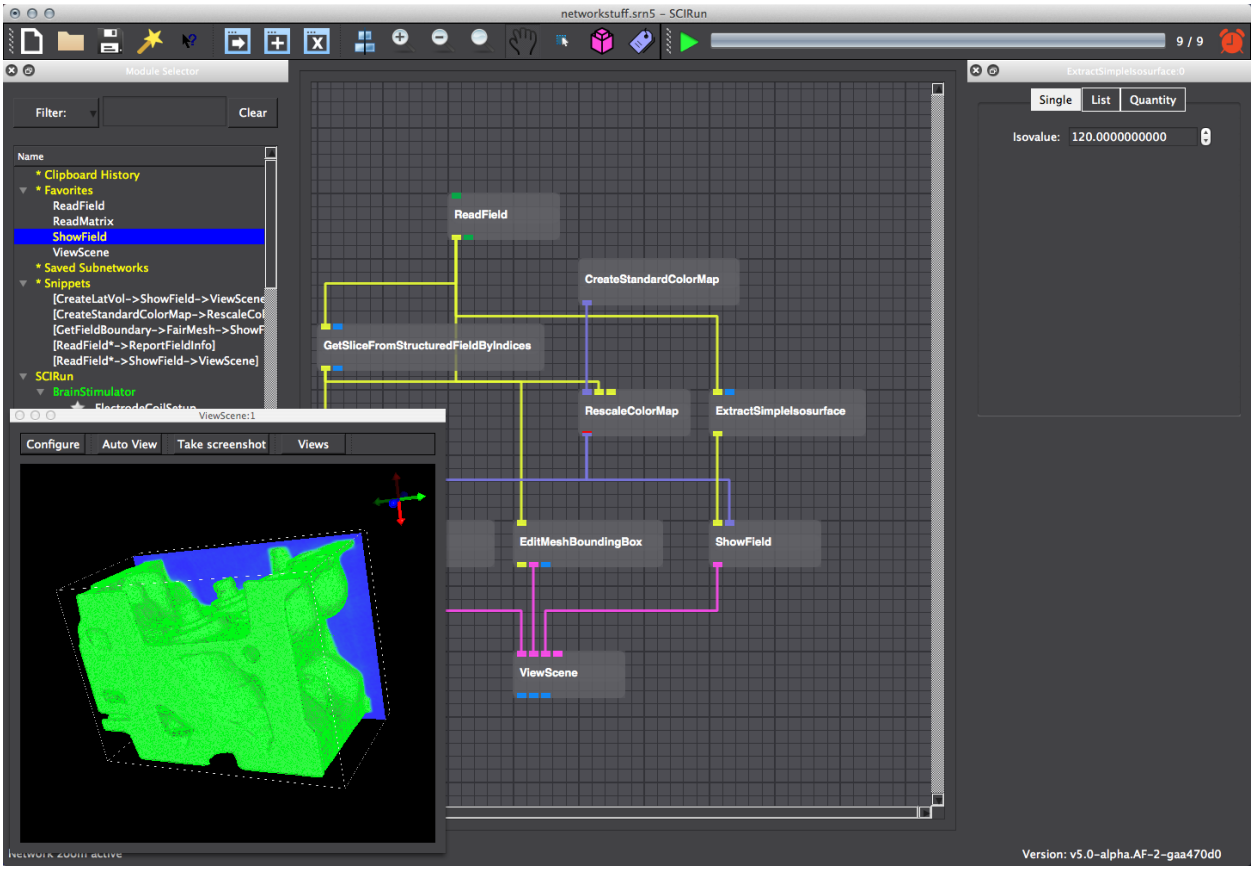


Fig. 2.12: Change the isovalue within ExtractSimpleIsosurface UI.

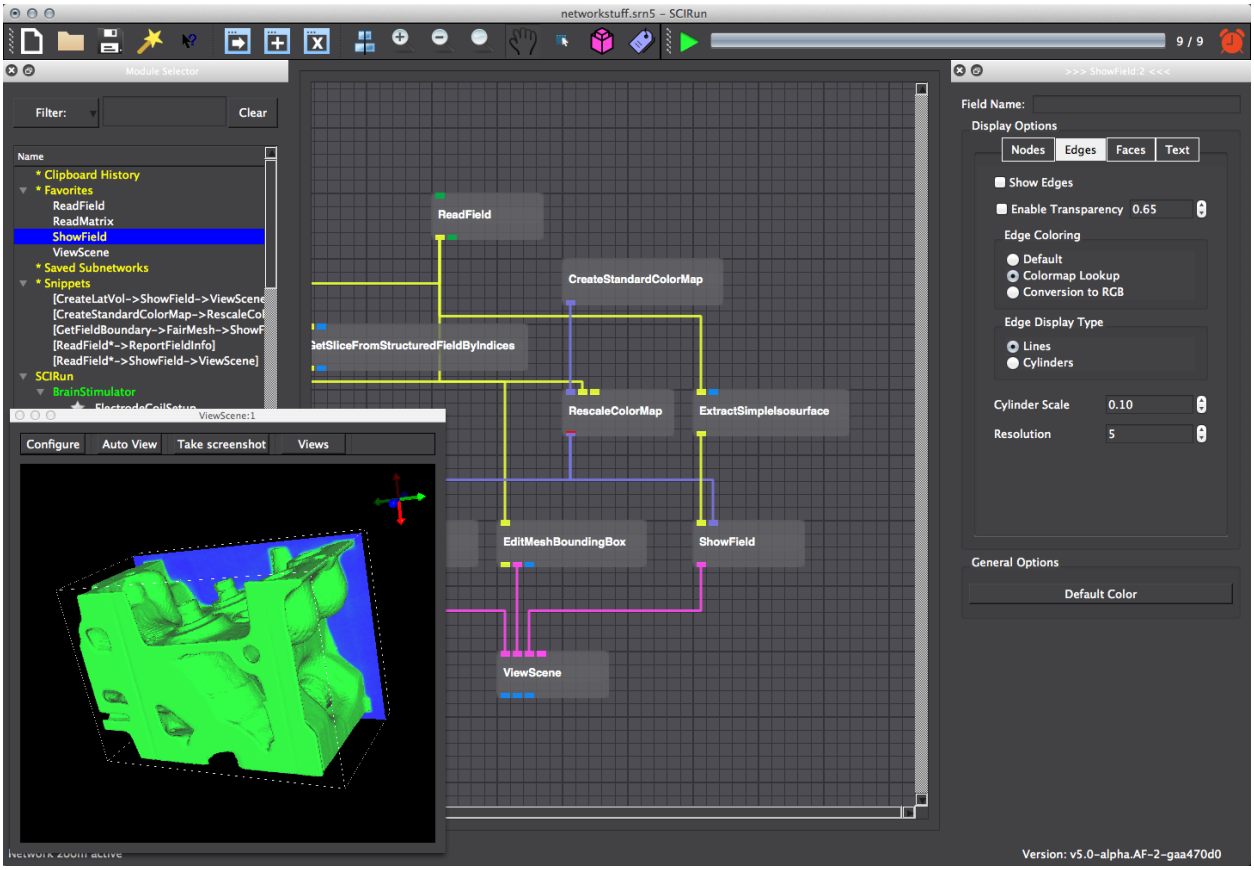


Fig. 2.13: Adjusting parameters within the ShowField UI helps to better visualize the isosurface.

2.3 Create, Manipulate and Visualize Field

2.3.1 Create Field

Create and manipulate a structured mesh type in this exercise. Start by creating a lattice volume using **CreateLatVol** module. Assign data at nodes using **CalculateFieldData** module. Connect CalculateFieldData to CreateLatVol. Input the following expression \(RESULT=sqrt(X*X+Y*Y+Z*Z) \) to compute data for each node within the CreateFieldData UI.

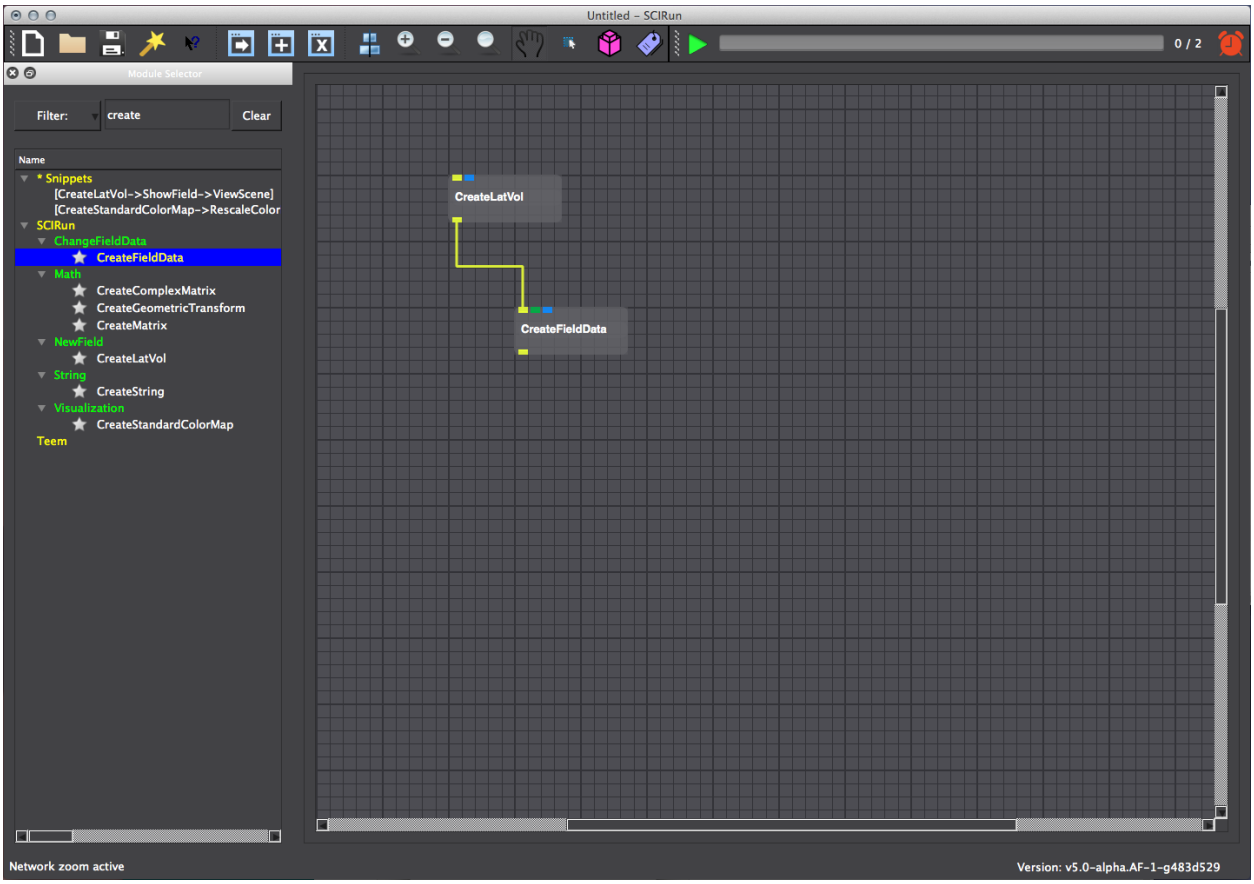


Fig. 2.14: Create lattice volume field using CreateLatVol module.

2.3.2 Isosurface

Generate the isosurface by instantiating and connecting an ExtractSimpleIsosurface module to CalculateFieldData (Fig. 2.16). Adjust the isovalue within the ExtractSimpleIsosurface UI so that the isosurface can be visualized (Fig. 2.17). Add a color map and visualize the isosurface as in the previous *example* (Fig. 2.18). Show the mesh bounding box as in *Show Bounding Box* section (Fig. 2.19).

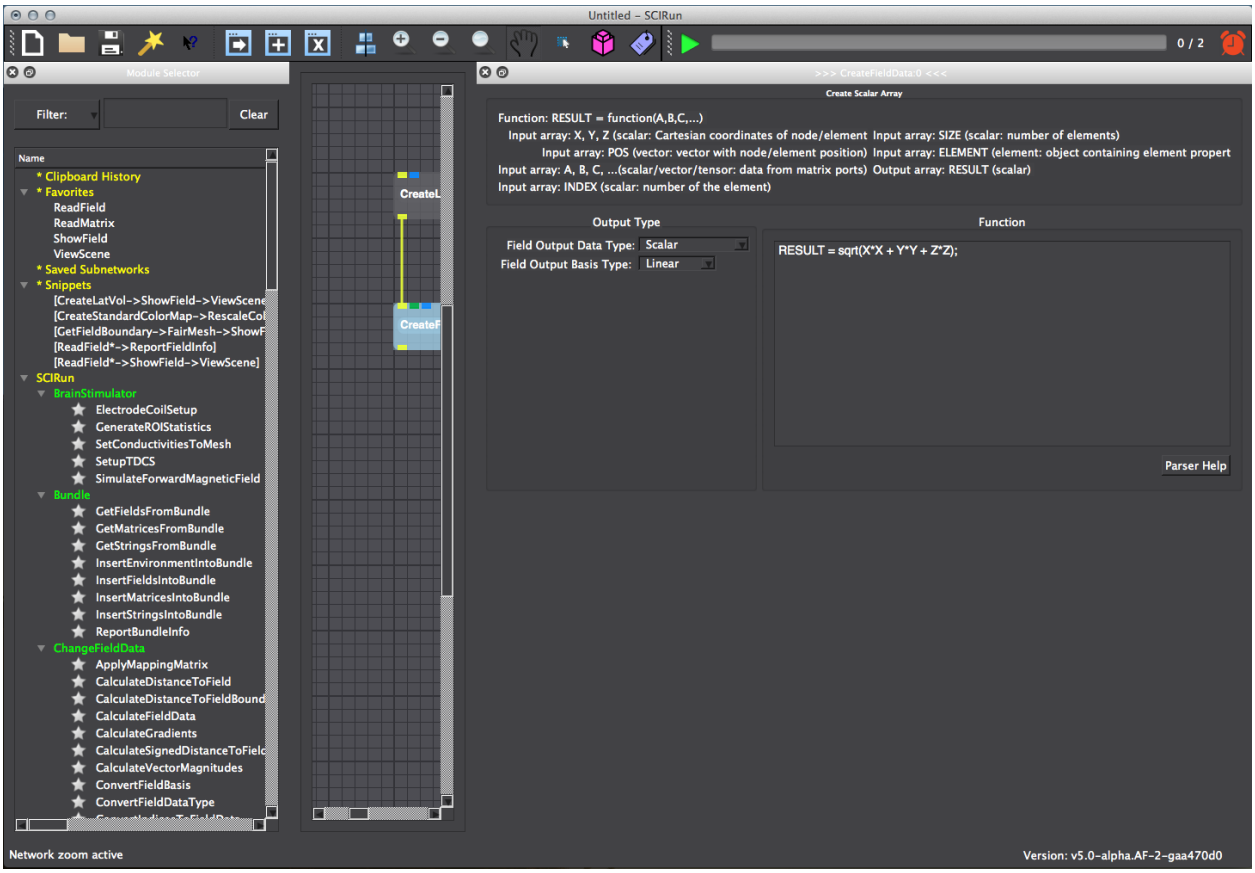


Fig. 2.15: Create a new field by inputting an expression into the CreateFieldData UI.

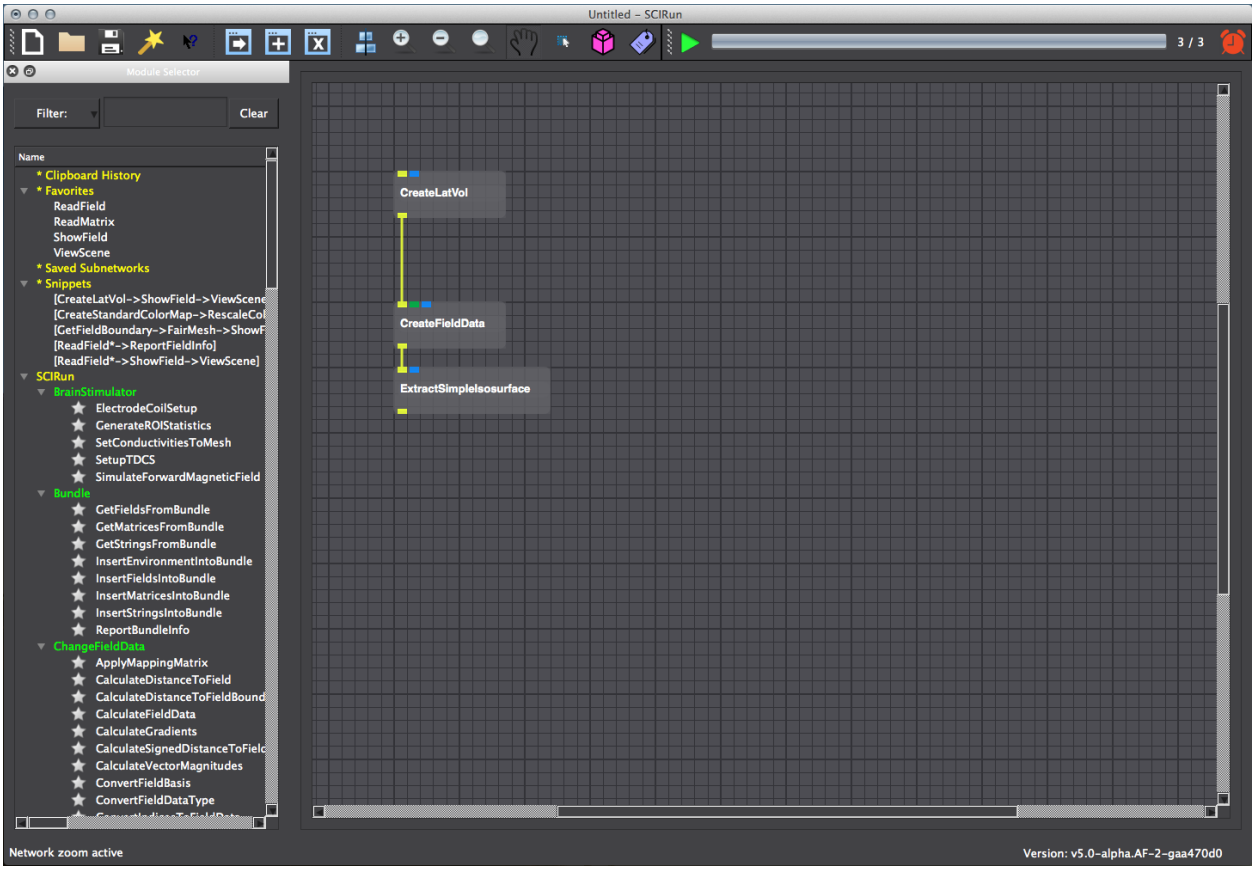


Fig. 2.16: Extract an isosurface from the field data.

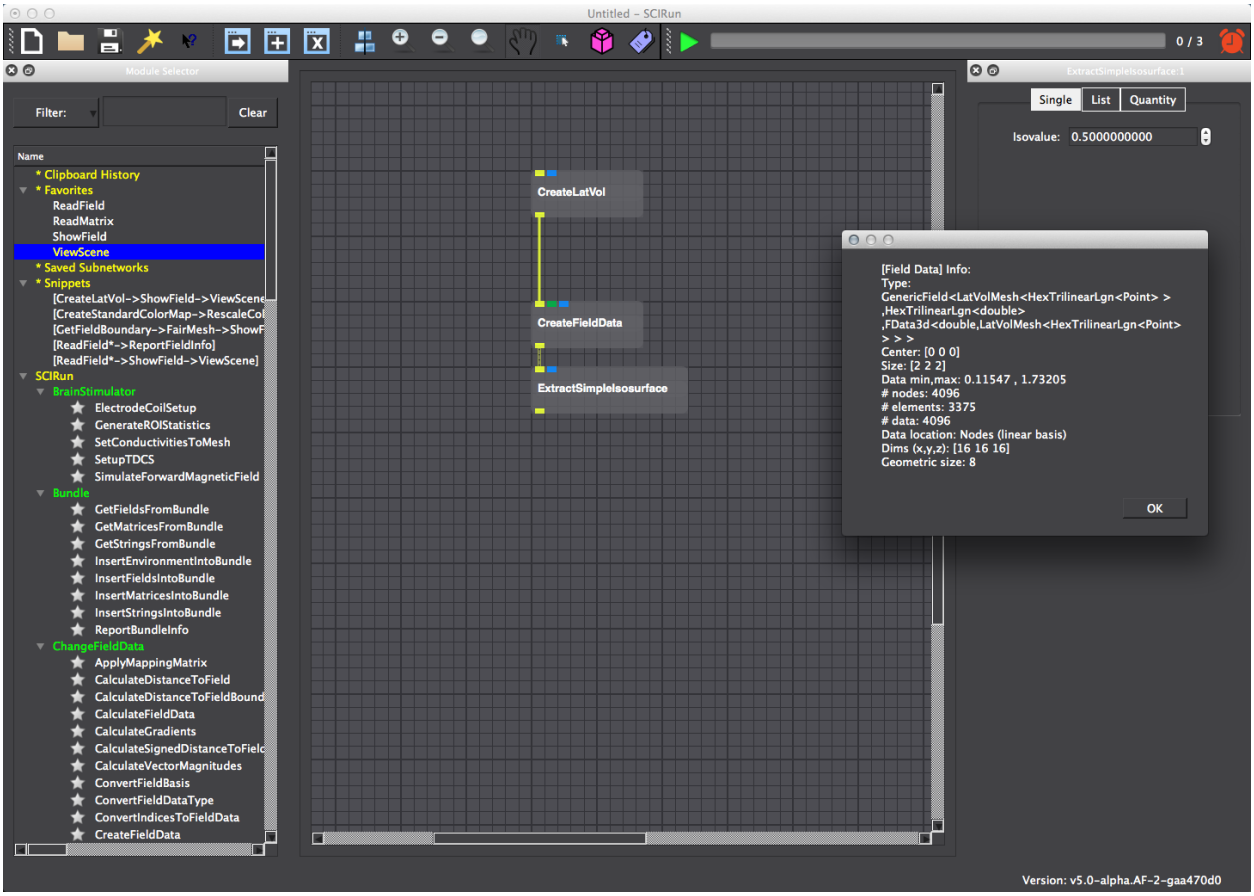


Fig. 2.17: Change the isovalue so that an isosurface can be visualized.

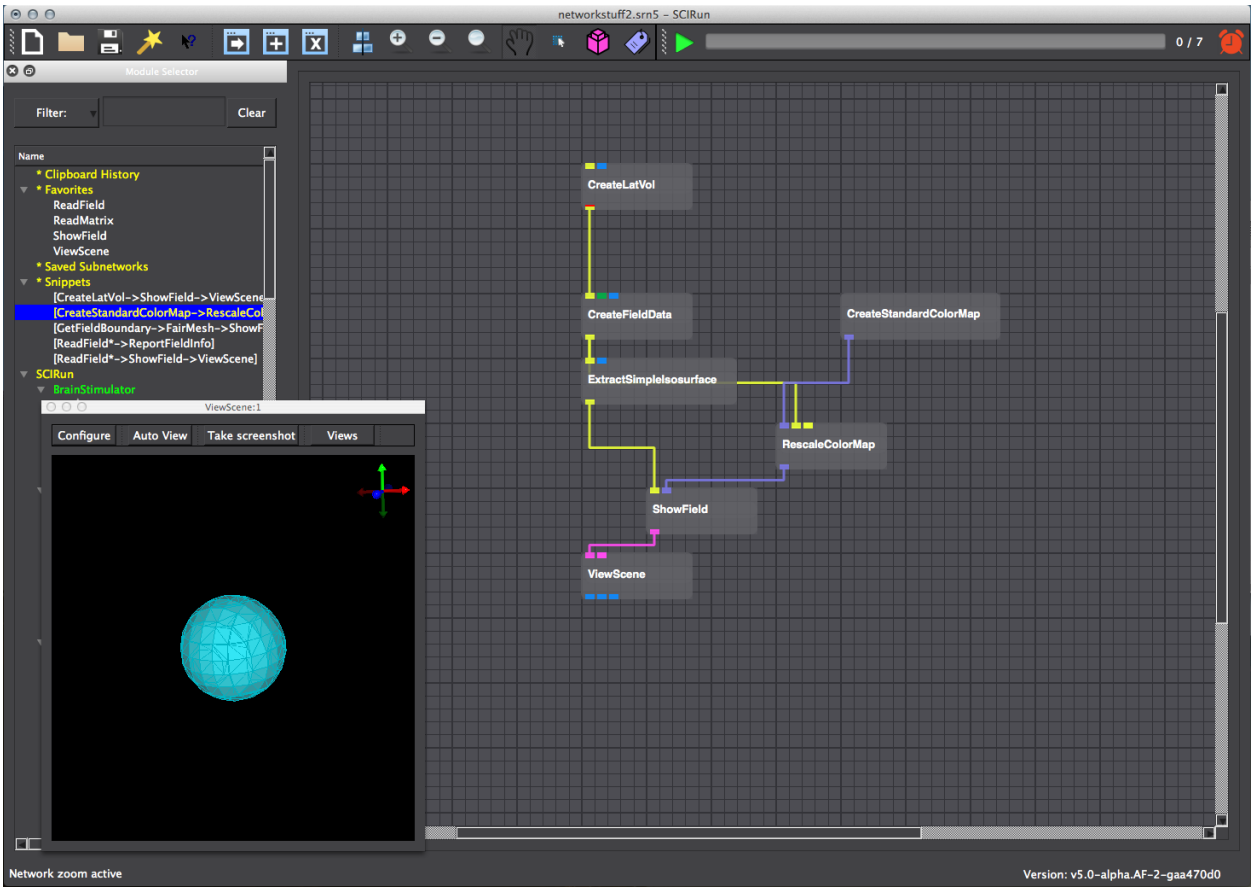


Fig. 2.18: Visualize the isosurface.

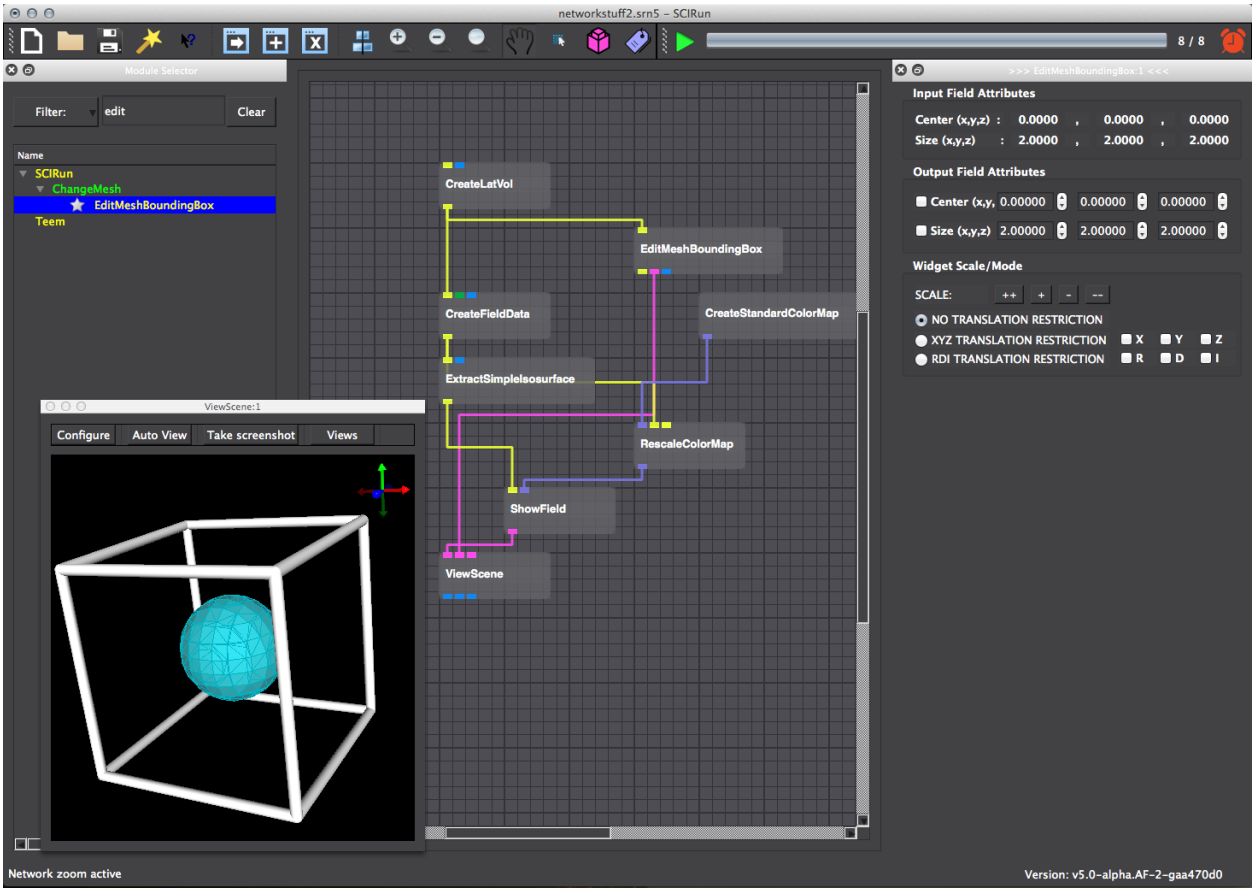


Fig. 2.19: Figure 3.6 Visualize the mesh's bounding box.

2.3.3 Slice Field

Extend the functionality of this network by slicing the field using GetSliceFromStructuredFieldByIndices as in previous *example*.

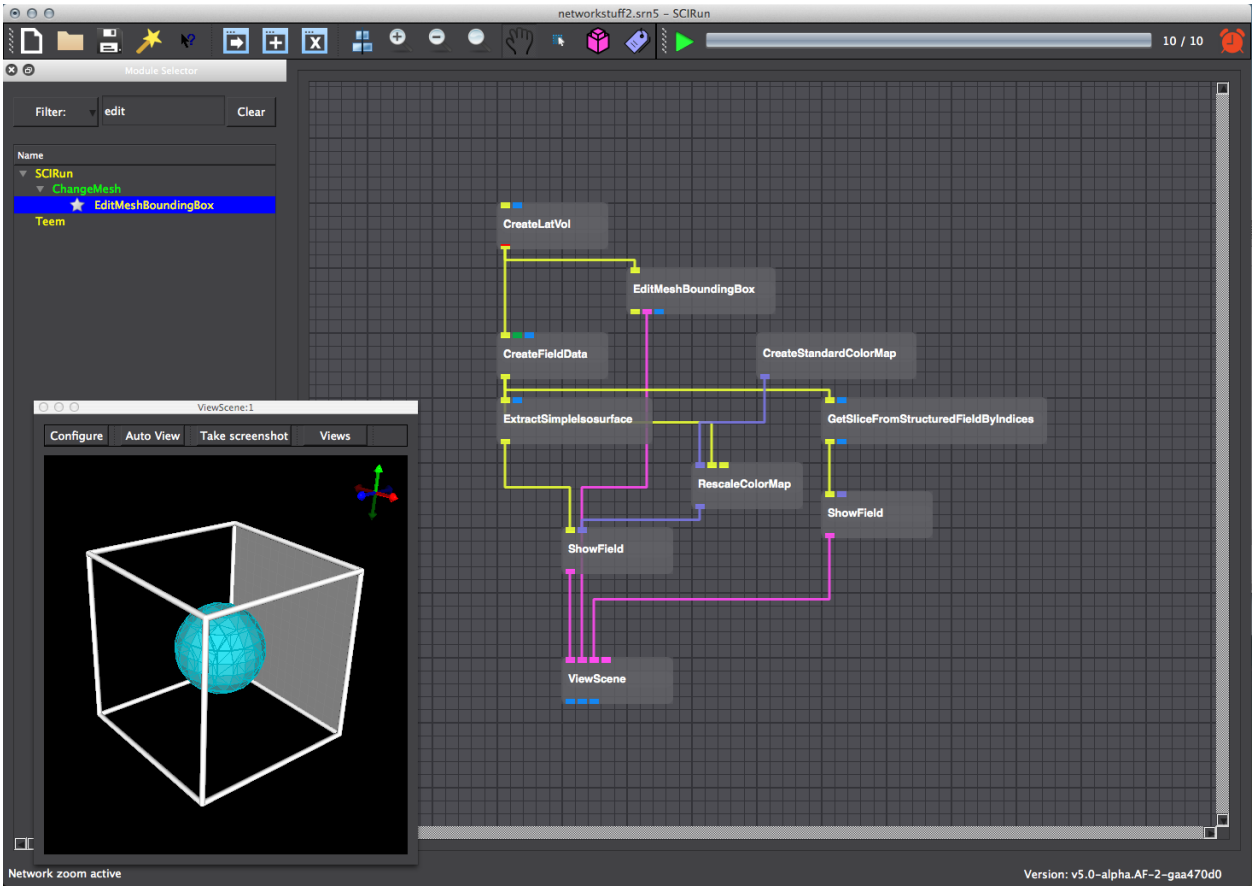


Fig. 2.20: Insert GetSliceFromStructuredFieldByIndices into the network.

2.3.4 Clip Field

Clip out a subset of the original field by converting the lattice volume to an unstructured mesh using **ConvertMeshToUnstructuredMesh** (Fig. 2.23) and adding **ClipFieldByFunction** (Fig. 2.24) to the network. Set the clipping location setting in ClipFieldByFunction to *all nodes*. Use the expression `DATA1>1 && X<0` to clip the field (Fig. 2.25).

Extract Boundary

At this point, it will be necessary to map the fields by interpolating the the boundary surface field to the clipping field. First, use **BuildMappingMatrix** to build a matrix that maps a linear combination of data values in the clipping field to a value in the boundary field. Then use **ApplyMappingMatrix** to multiply the data vector of the clipping field with the mapping matrix to obtain the data vector for the boundary surface field (Fig. 2.26). Use GetFieldBoundary to extract the boundary surface from the lattice volume and use it as input into the ApplyMappingMatrixModule and BuildMapping Matrix (Fig. 2.27). Port the output from the BuildMappingMatrix module to ApplyMappingMatrix and visualize the resultant field using a ShowFieldModule (Fig. 2.28). Add a colormap to and enable transparency in ShowField UI for further functionality (Fig. 2.29).

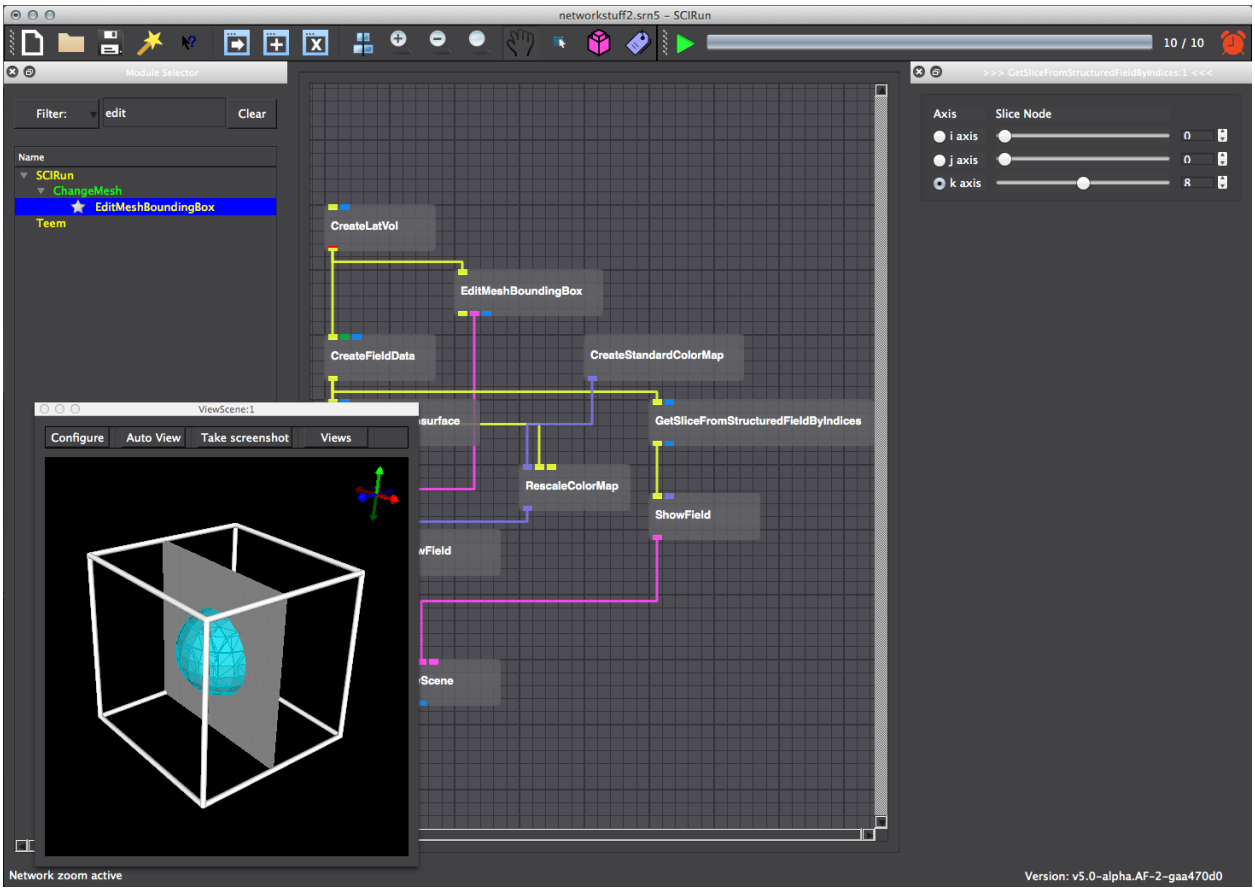


Fig. 2.21: Change the slice index using the GetSliceFromStructuredFieldByIndices UI.

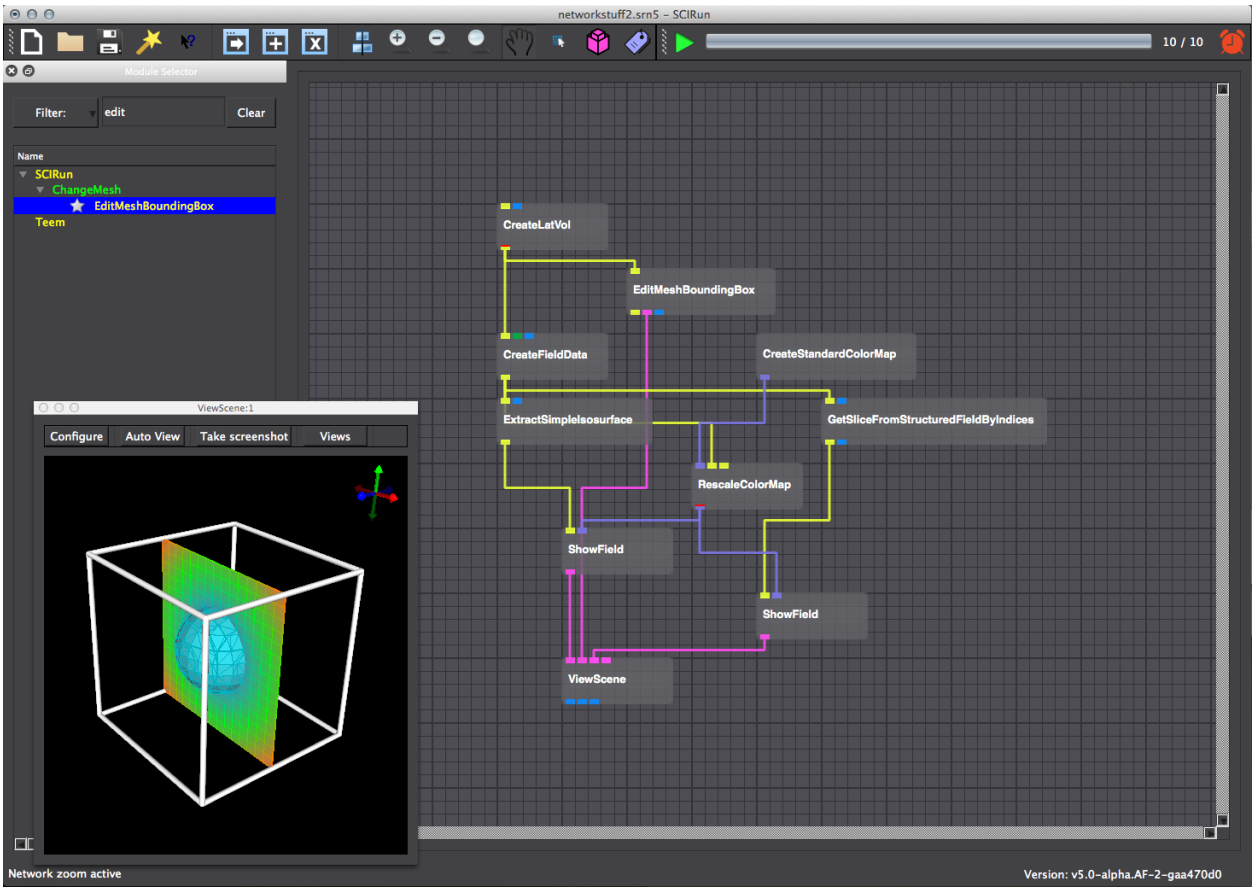


Fig. 2.22: Attach the RescaleColorMap module to the ShowField module.

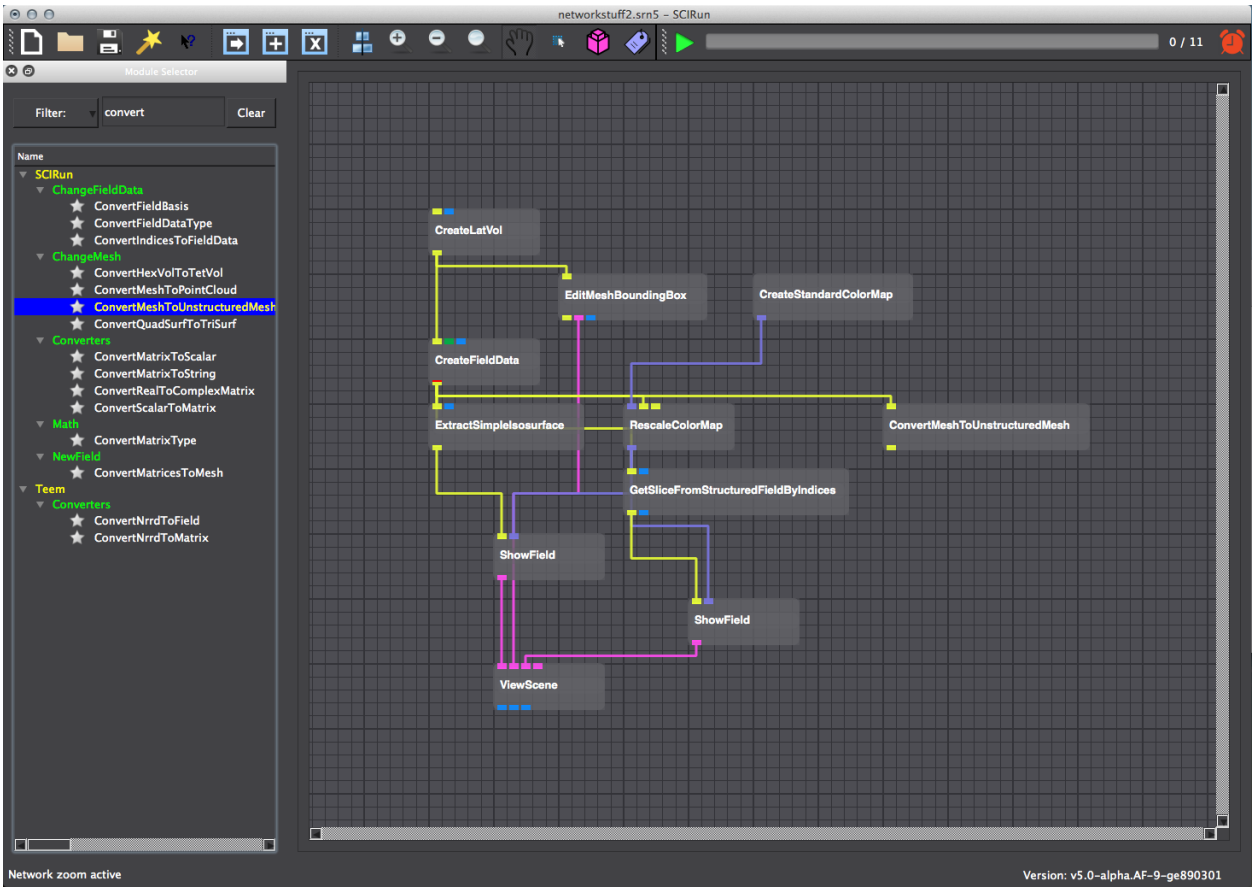


Fig. 2.23: Convert the original field to an unstructured mesh.

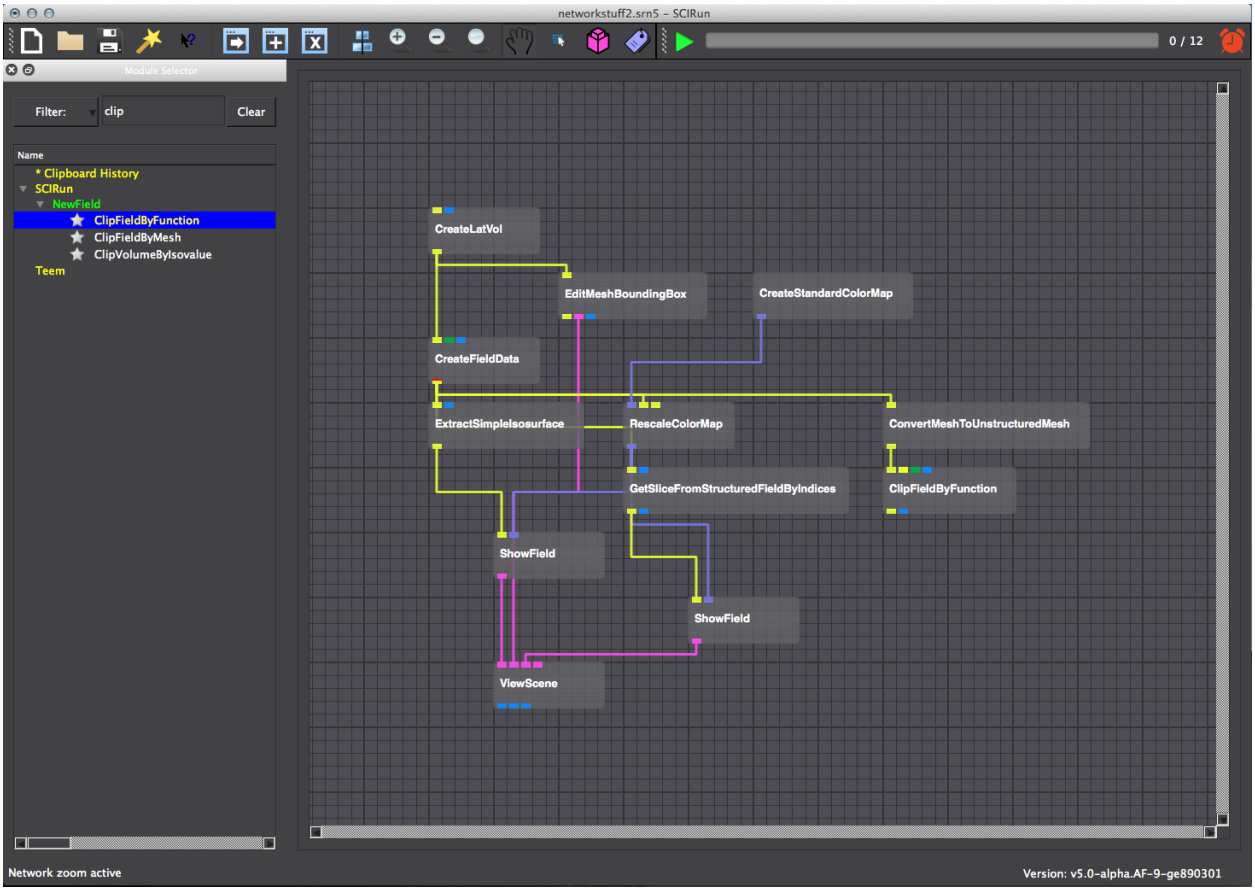


Fig. 2.24: Insert a ClipFieldbyFunction module.

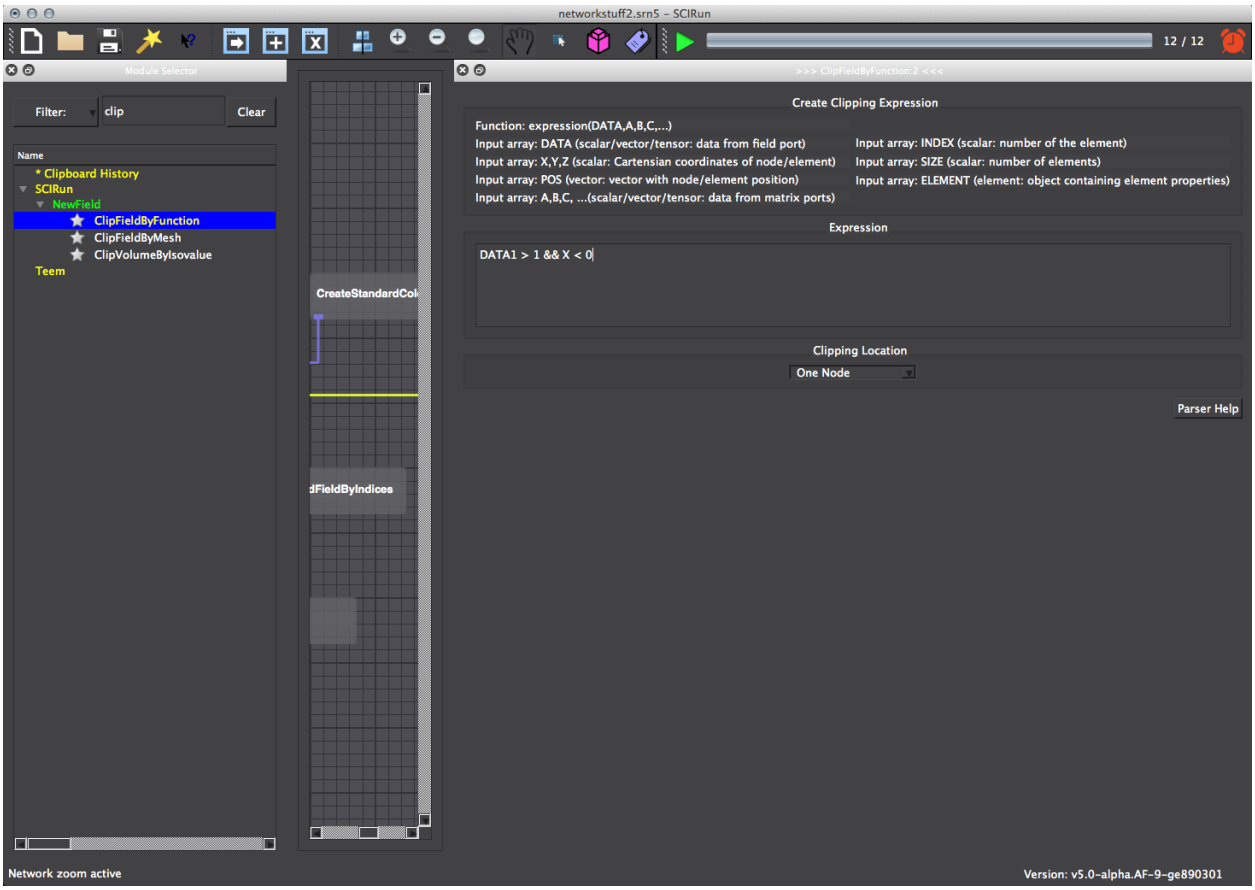


Fig. 2.25: Clip the field by entering an expression in the ClipField UI.

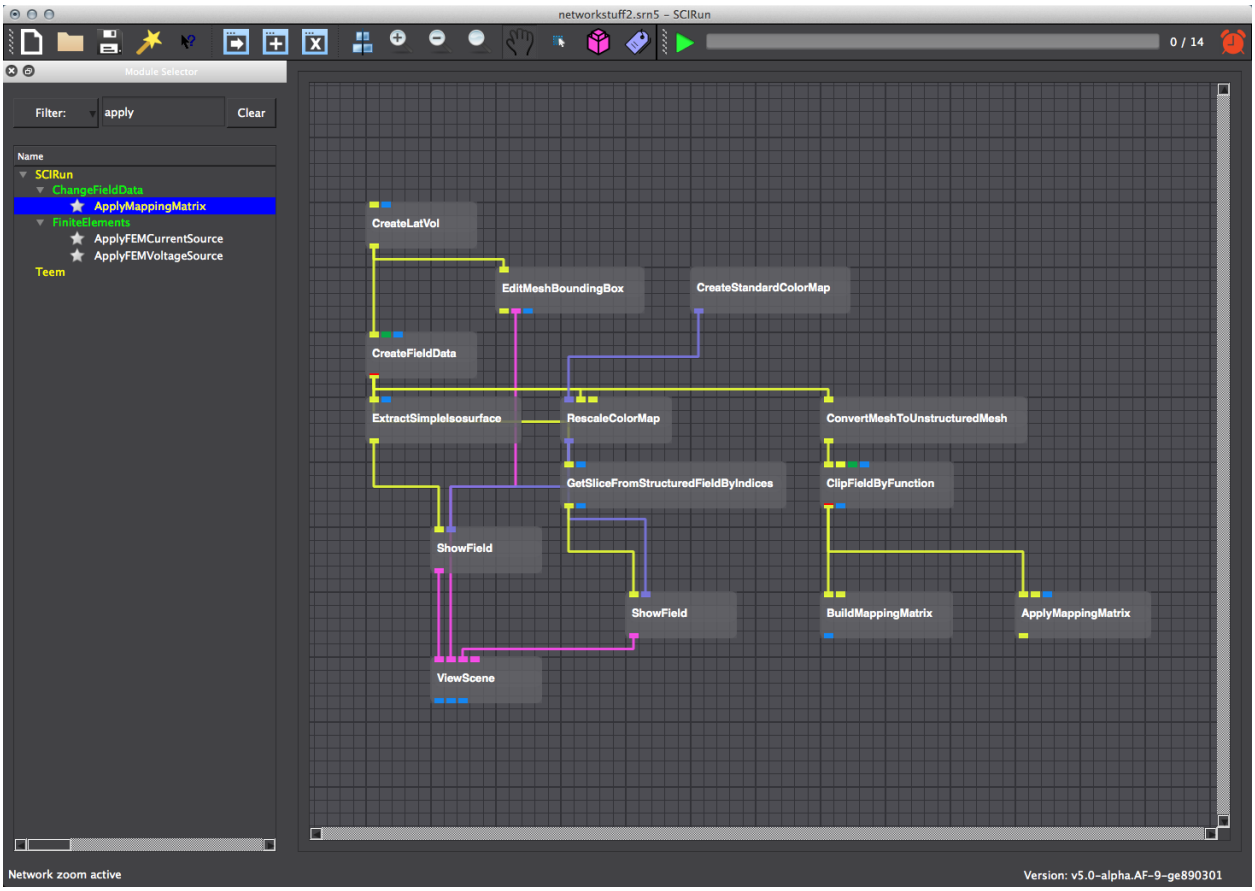


Fig. 2.26: Build and apply the mapping network connections.

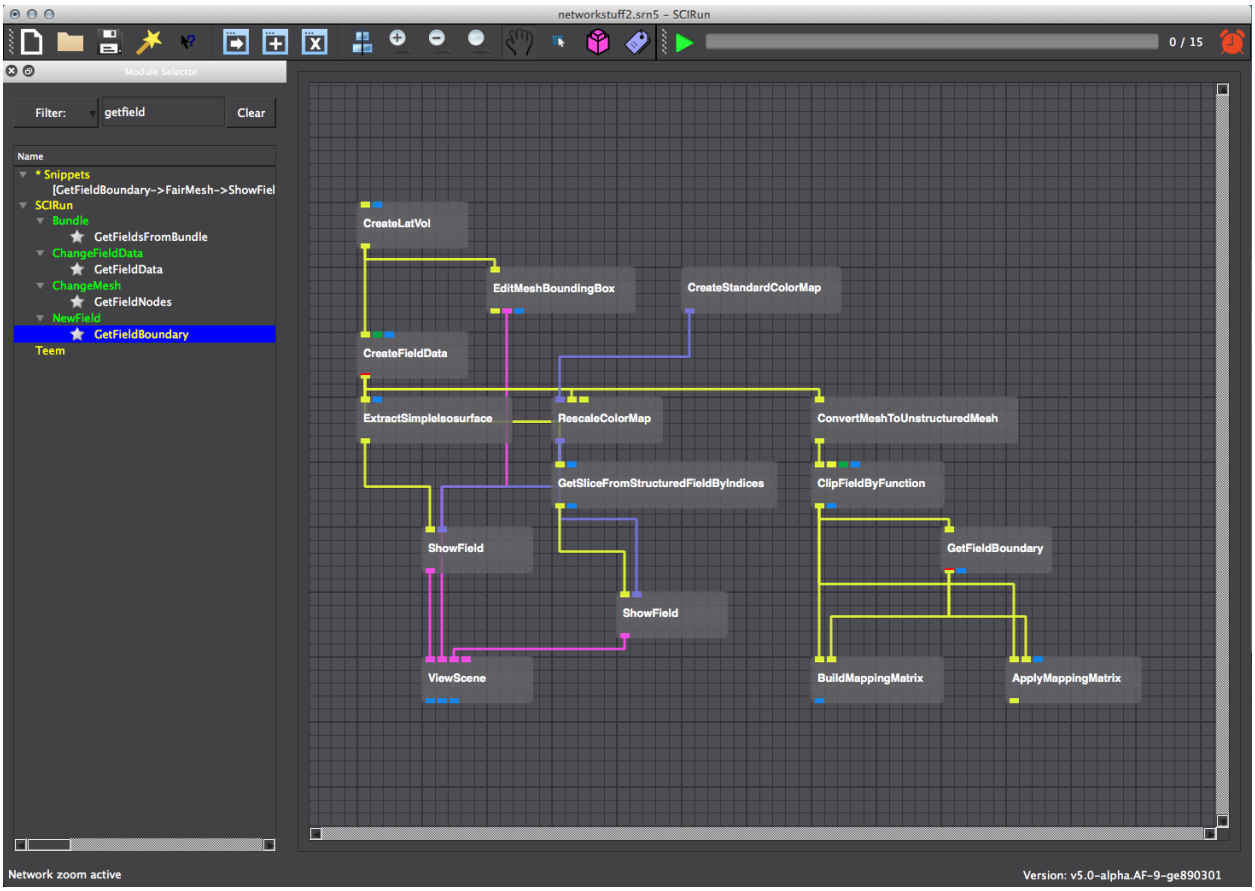


Fig. 2.27: Add GetFieldBoundary to the network.

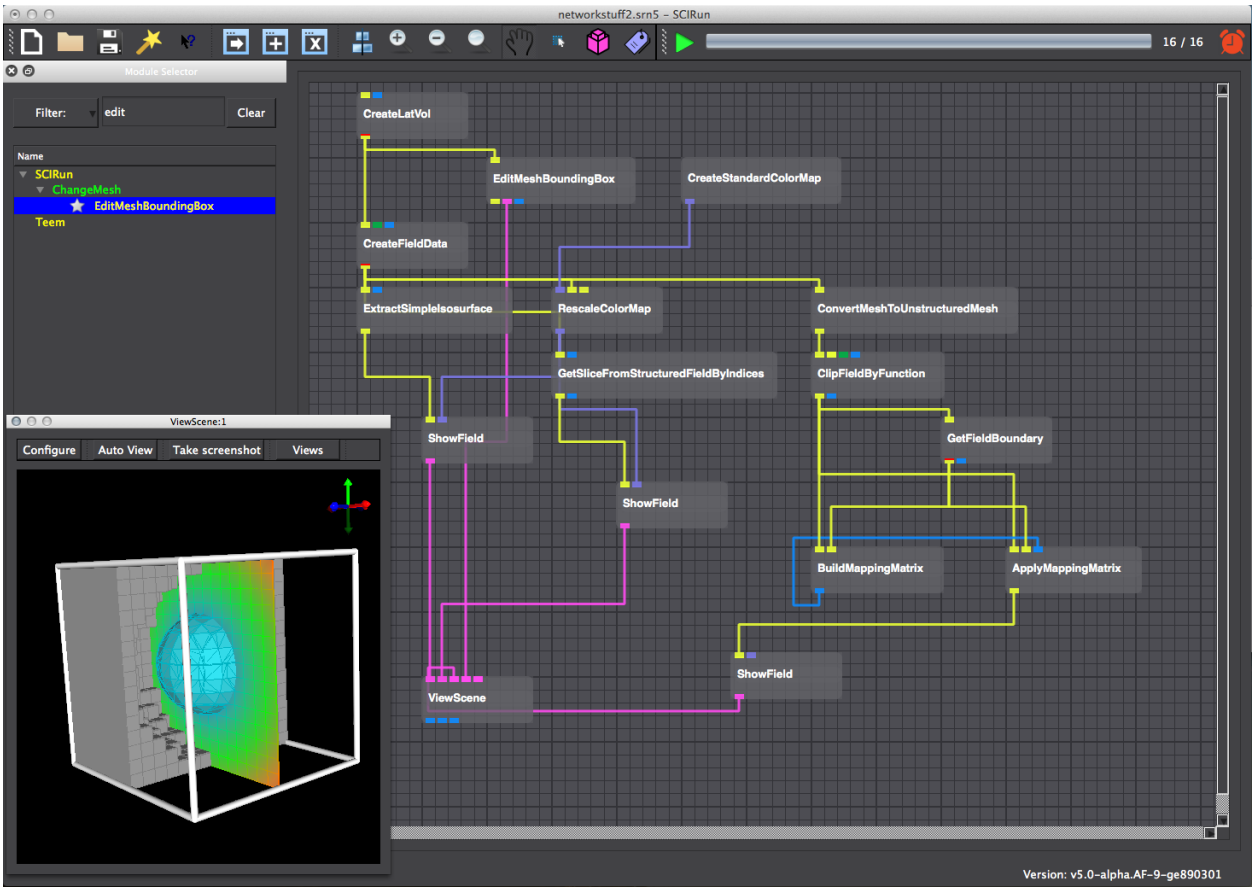


Fig. 2.28: Connect all the modules for mapping and visualize the output.

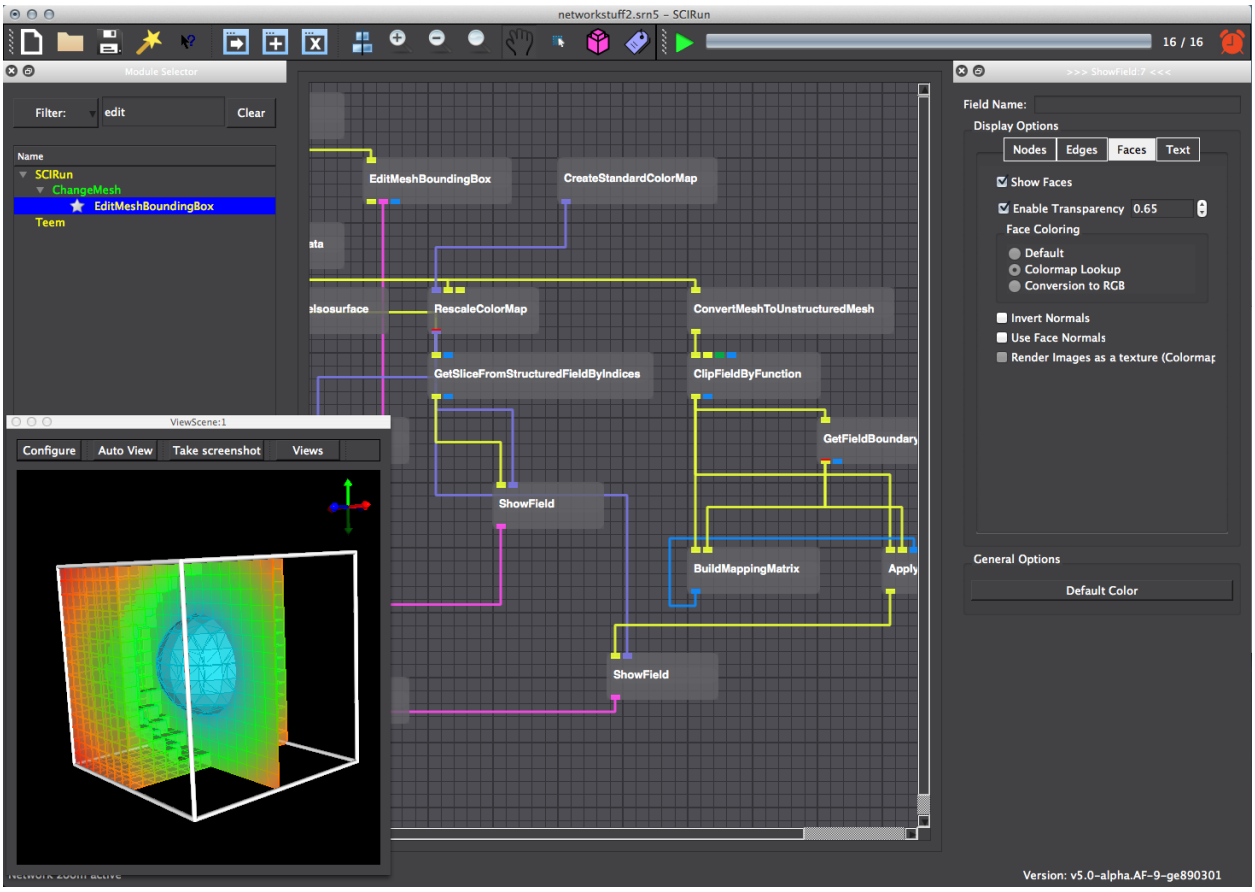


Fig. 2.29: Add a colormap and enable transparency.

Finally, it is not strictly necessary to explicitly convert the original mesh to an unstructured mesh using ConvertMeshToUnstructuredMesh because ClipFieldByFunction can implicitly convert structured mesh types to unstructured mesh types before clipping the field. As a final exercise, delete ConvertMeshToUnstructuredMesh from the network and try to obtain the same result.

CHAPTER

THREE

CLASSIC TUTORIAL

3.1 Overview

SCIRun is a modular dataflow programming Problem Solving Environment (PSE). SCIRun has a set of Modules that perform specific functions on a data stream. In SCIRun, a module is represented by a rectangular box on the Network Editor canvas. Each module reads data from its input ports, calculates the data, and sends new data from output ports. Data flowing between modules is represented by pipes connecting the modules. A group of connected modules is called a Dataflow Network (see Fig. 3.3), and are saved as `.srn5` files. Any number of nets can be created, each solving a separate problem.

This tutorial demonstrates the use of SCIRun to visualize a tetrahedral mesh and the construction of a network comprised of three standard modules: ReadField, ShowField, and ViewScene. This tutorial also instructs the user on reading Field data from a file, setting rendering properties for the nodes, edges, and faces (**the nodes are rendered as blue spheres**), and rendering geometry to the screen in an interactive ViewScene window).

3.1.1 Software requirements

SCIRun

SCIRun is available for download on the GitHub release page. Make sure to update to the most up-to-date release available, which will include the latest bug fixes. Use the supplied installers to install SCIRun on your local machine. Linux users may have to *build from source*.

Download the SCIRunData file as a zip or tgz. Unpack it in a convenient location, then add the base directory to the SCIRun Data Path in the SCIRun preference window, under the Paths tab (Fig. 3.1)

3.2 Starting SCIRun

To open the main SCIRun window, launch SCIRun by either double clicking on the binary icon or by launching SCIRun from the command line. Once SCIRun has been launched, the main SCIRun network editor will appear (Fig. 3.2).

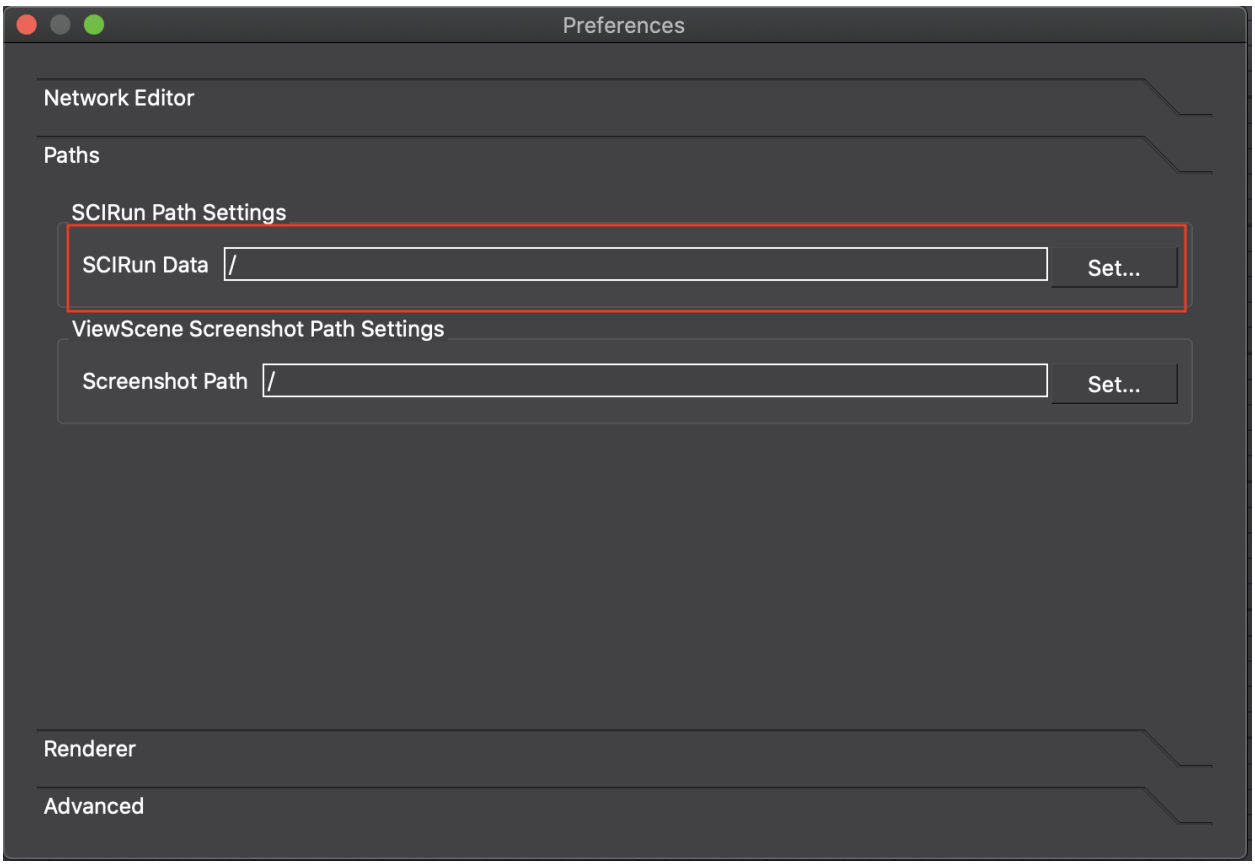


Fig. 3.1: The the Paths tab of the SCIRun preference window.

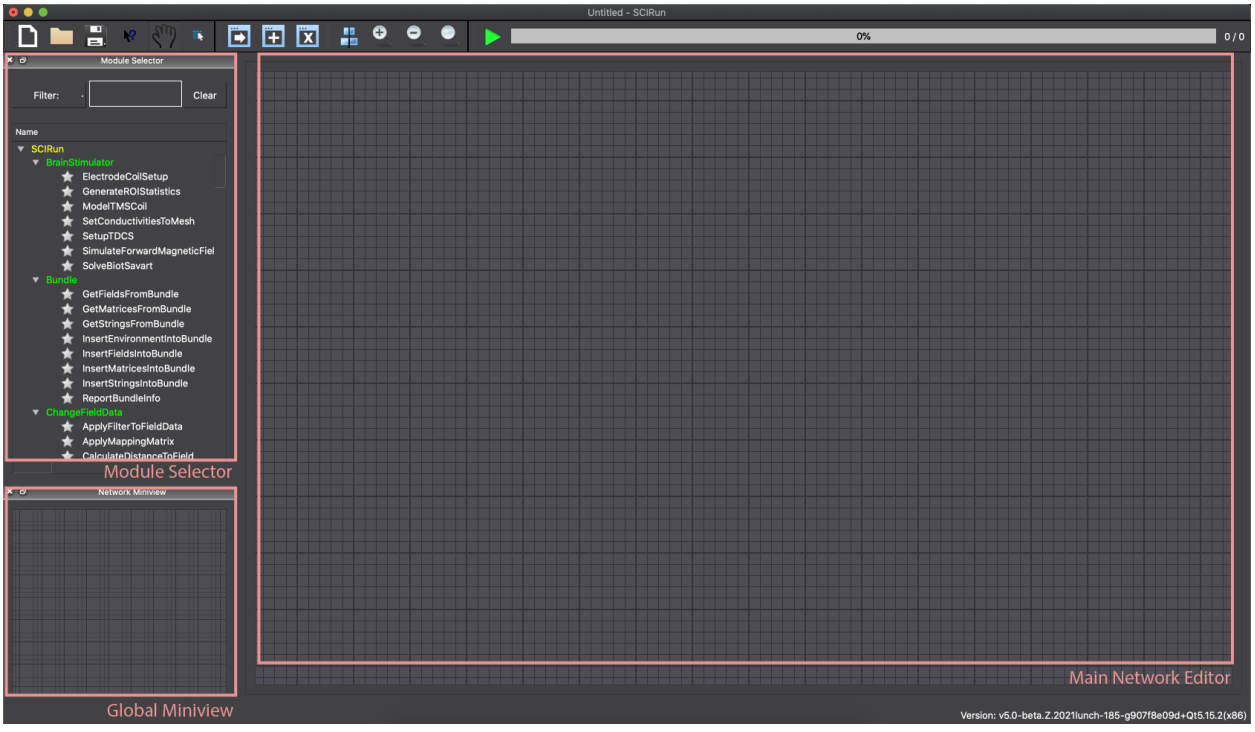


Fig. 3.2: The main SCIRun window and components

3.2.1 SCIRun Network Building Blocks: Modules and Connections

Lets begin constructing the network pictured in (Fig. 3.3). (Note: subsequent tutorial chapters expand on this network, adding more features and functionality.) This network loads a geometric mesh from a data file and renders it to the screen.

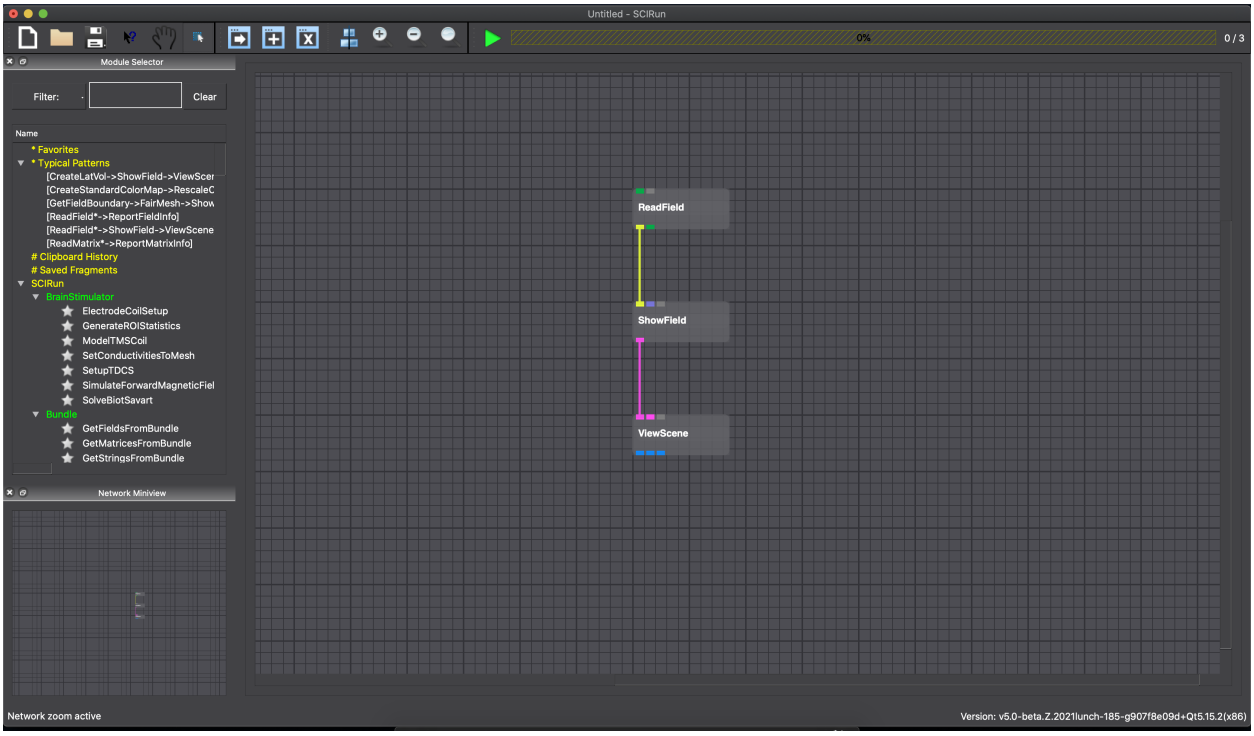


Fig. 3.3: Network editor after building example network

Modules: A module is a single-purpose unit that functions within a dataflow environment. Modules have at least one input port for receiving data, located at the top of the module, or one output port for sending data, located at the bottom of the module (Fig. 3.4).

All modules have an indicator that alerts the user to messages that exist in a module's log. Different colors represent different types of messages. Gray means no message, blue represents a Remark, yellow a Warning, and red an Error. To read messages, click the module's indicator button to open the log window.

Pipes: Data is transferred from one module to another using dataflow connections, commonly referred to as Pipes. Each dataflow pipe transfers a specific datatype in SCIRun, denoted by a unique color. Pipes run from the output Port of one module to the input Port(s) of one or more other modules. Ports of the same color correspond to the same datatype and can be connected.

Two or more connected modules form a SCIRun network.

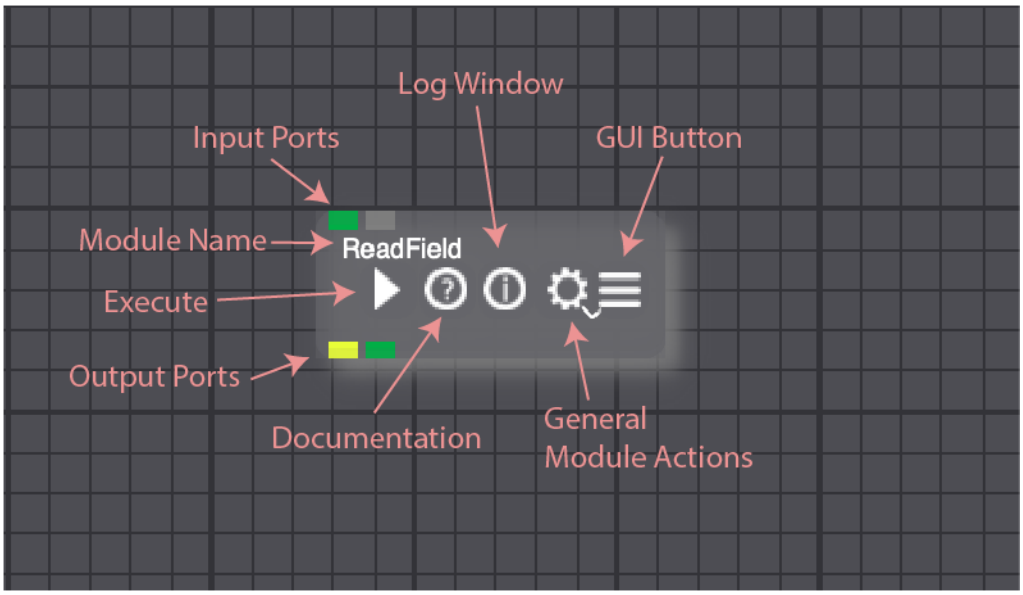


Fig. 3.4: SCIRun Module icon

3.3 ReadField Module

Now it is time to begin creating a SCIRun network. First, create a **ReadField** module, which will be used to load a SCIRun Field dataset from disk.

Select **ReadField** under the DataIO section in the module selector located on the left side the the main SCIRun window, as show in (Fig. 3.5).

The **ReadField** module will appear on the NetEdit frame. Now, set user interface parameters for this module, by pressing the UI button on the module. This brings up a standard file selection dialog (Fig. 3.6). Select the utahtorso-lowres/utahtorso-lowres-voltage.tvd.fld input file. (Note: This file can be found in the SCIRunData directory. This directory should have been downloaded and installed when SCIRun was installed.) This dataset contains a low resolution tetrahedral mesh of a human torso.

Once the file has been selected, the following will occur:

- The file selection window disappears.
- The module reads in the dataset (a SCIRun Field) and the progress bar turns green.

3.3.1 Brief Field Overview

A Field contains a geometric mesh, and a collection of data values mapped on to the mesh. Data can be stored at the nodes, edges, faces, and/or cells of the mesh. In this case, a tetrahedral mesh with voltages defined at the nodes of the mesh has been selected.

The dimensionality of the mesh type determines the available storage locations. For example, a TriSurf mesh has nodes, edges, and planar faces, but not cells, which are assumed to be three-dimensional elements. As a result, a TriSurf cannot store data in cells, but can store data in edges or faces.

See *Appendix 1* for a description of various types of geometric meshes, data values, and mappings SCIRun supports.

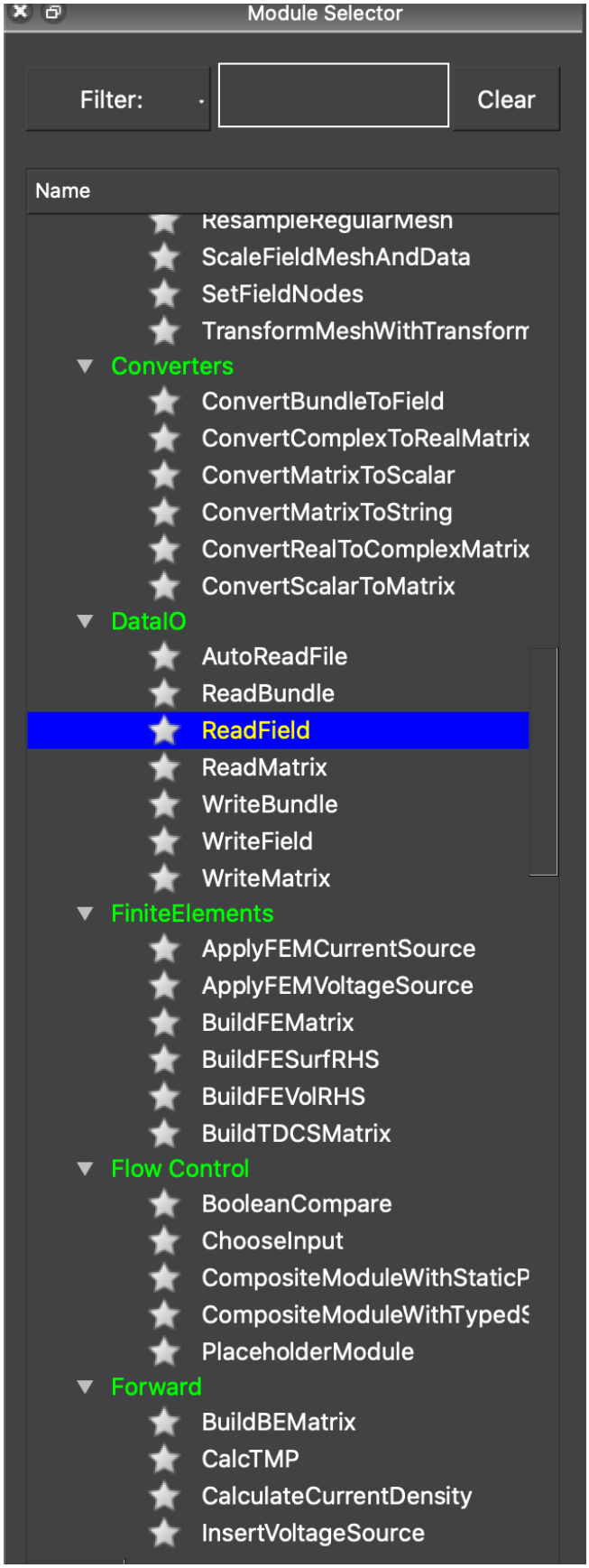


Fig. 3.5: Module creation (ReadField)

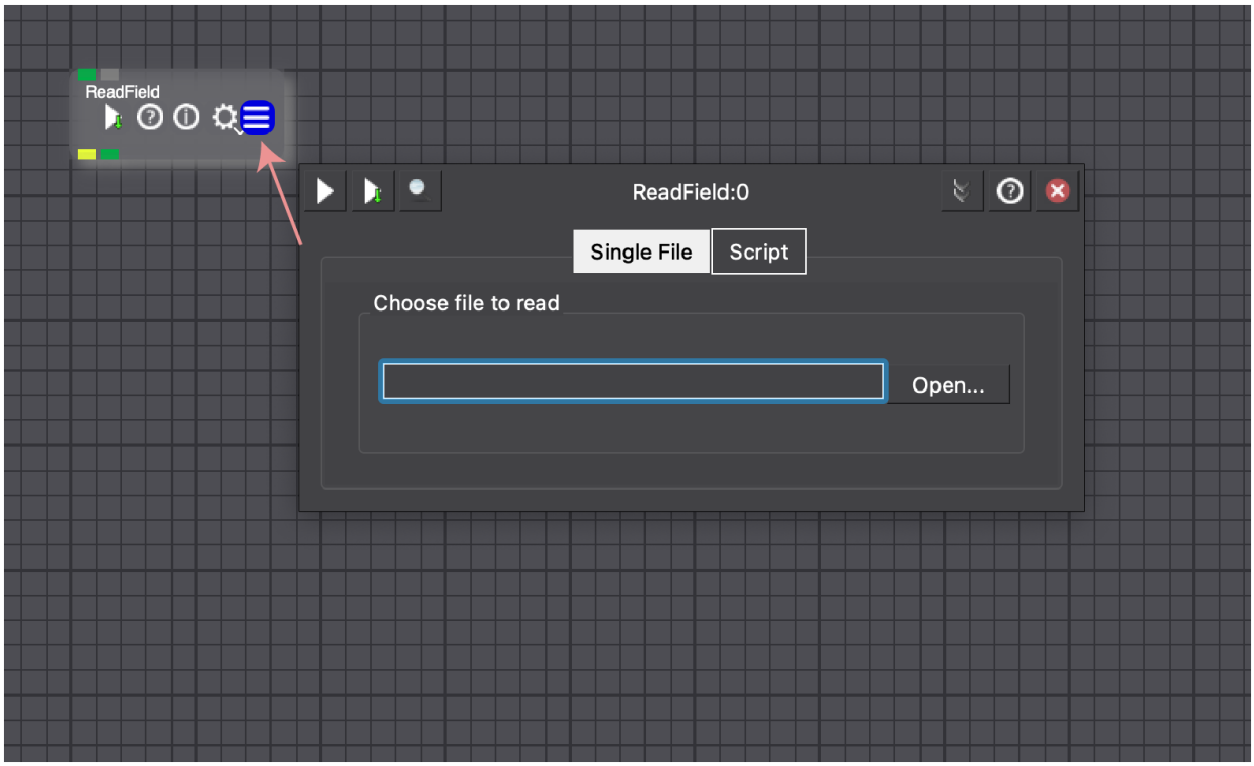


Fig. 3.6: ReadField file selection GUI

3.4 Hooking Modules Together

Now add a second module to the network. This module is used to visualize various Field types. Then connect the two modules in the canvas so data can flow between them.

1. Create a **ShowField** module using the DataIO section in the module selector (use the same menus used to create the ReadField module).
2. Position the mouse pointer over the yellow output port on the **ReadField**. Press and hold the left mouse button. The name of the port and lines indicating possible data pipe connections will appear.
3. Continue to hold the left mouse button and drag the mouse toward the first yellow **ShowField** input port.
4. The line turns red, showing the desired connection has been selected. See (Fig. 3.7).
5. Release the mouse button. A yellow pipe showing a data flow connection between **ReadField** and **ShowField** will appear (Fig. 3.8).

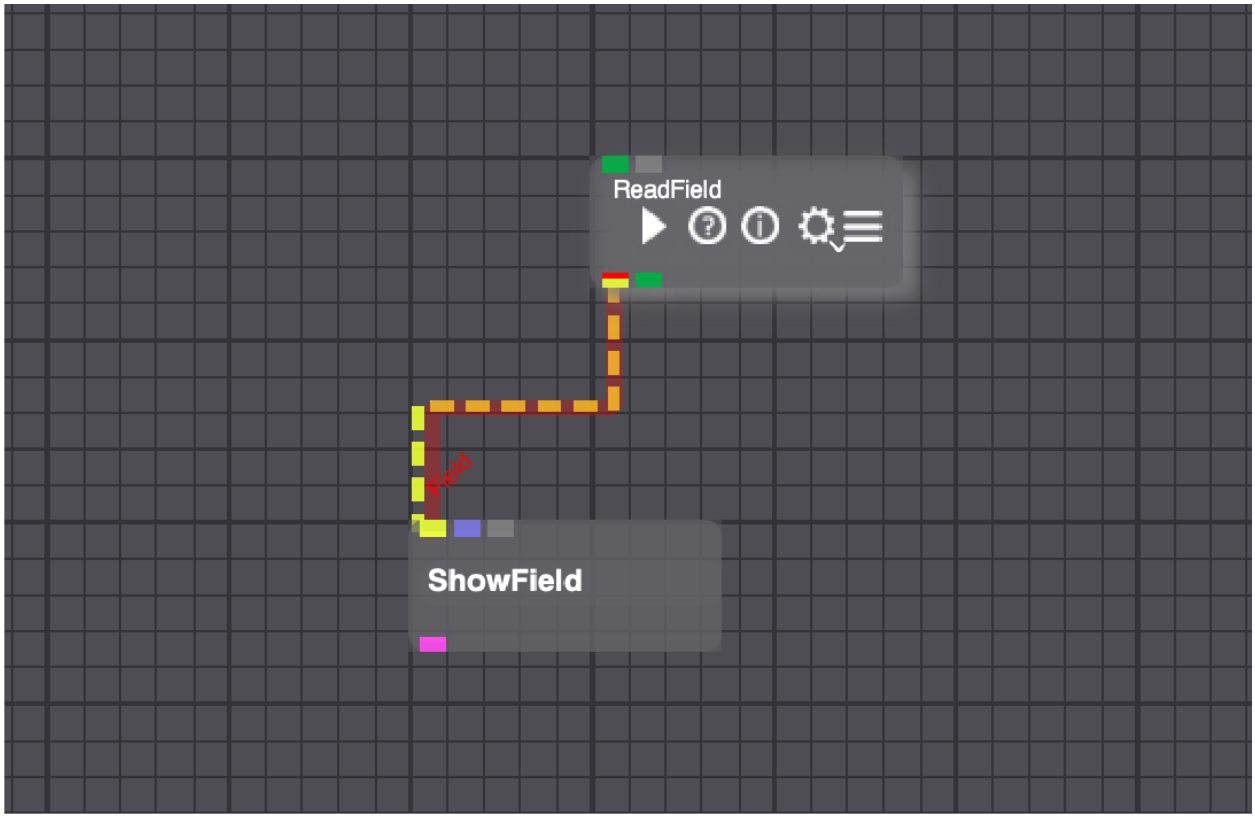


Fig. 3.7: Pipe Selection Options

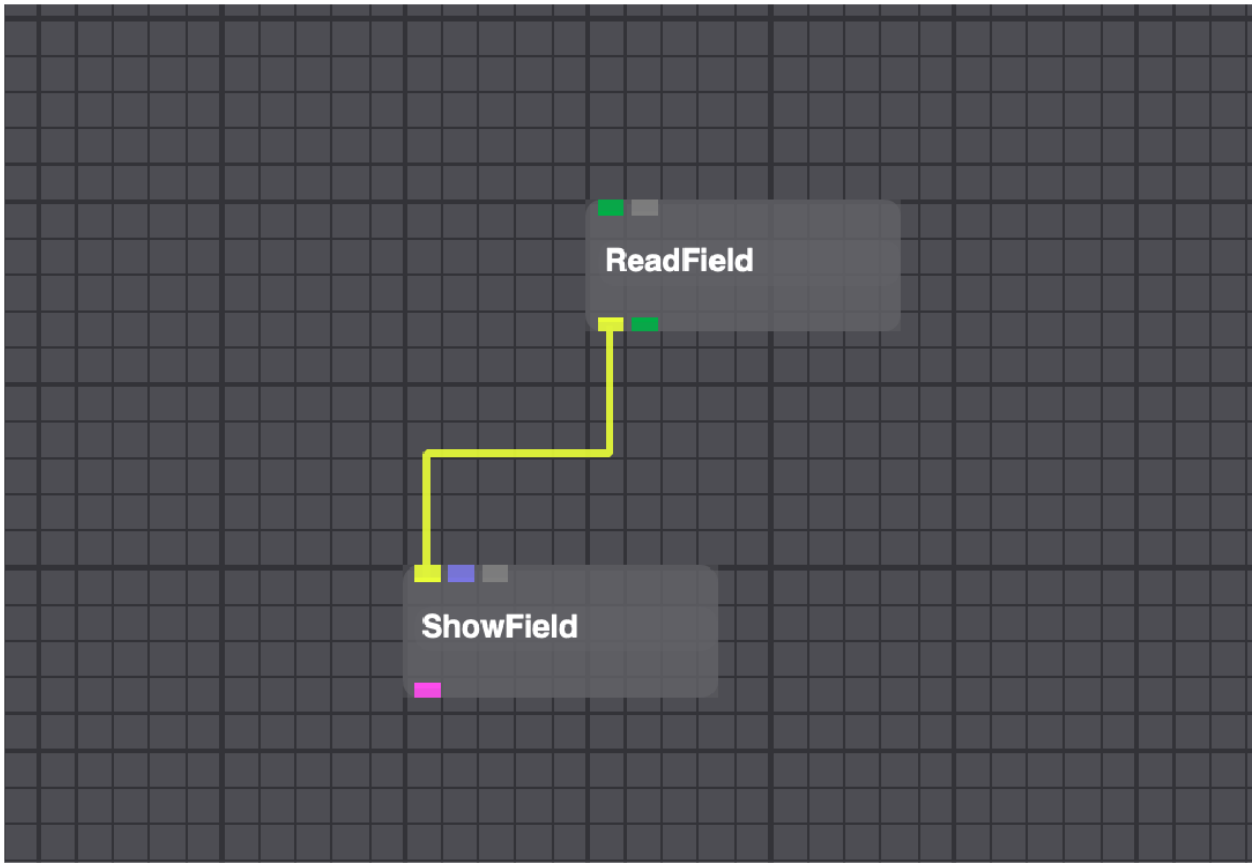


Fig. 3.8: Connected Dataflow Pipe

3.5 Setting the ShowField User INTERFACE_MODULES_

The **ShowField** module has options for changing the visual representations of a Field's geometry. To illustrate the module's functionality, change **ShowField** parameters using its GUI (found by clicking the GUI button on the **ShowField** module (Fig. 3.4)). Specifically, change the color of the nodes to blue spheres.

1. Select the UI button on the **ShowField** module.
2. Select the Default Color button near the top of the GUI to change the default color and a separate Color Chooser GUI appears.
3. In the Color Chooser GUI, use the sliders to adjust the color values. Select a blue color.
4. Select the OK button in the Color Chooser GUI. Notice that the Default Color swatch in the ShowField GUI has changed to blue. SCIRun and the ShowField GUI should now look like Fig. 3.9.
5. Close the Color Chooser GUI.
6. Set the name of the Field to Voltage. This makes it easy to identify the Field in the **ViewScene** Window (discussed later).

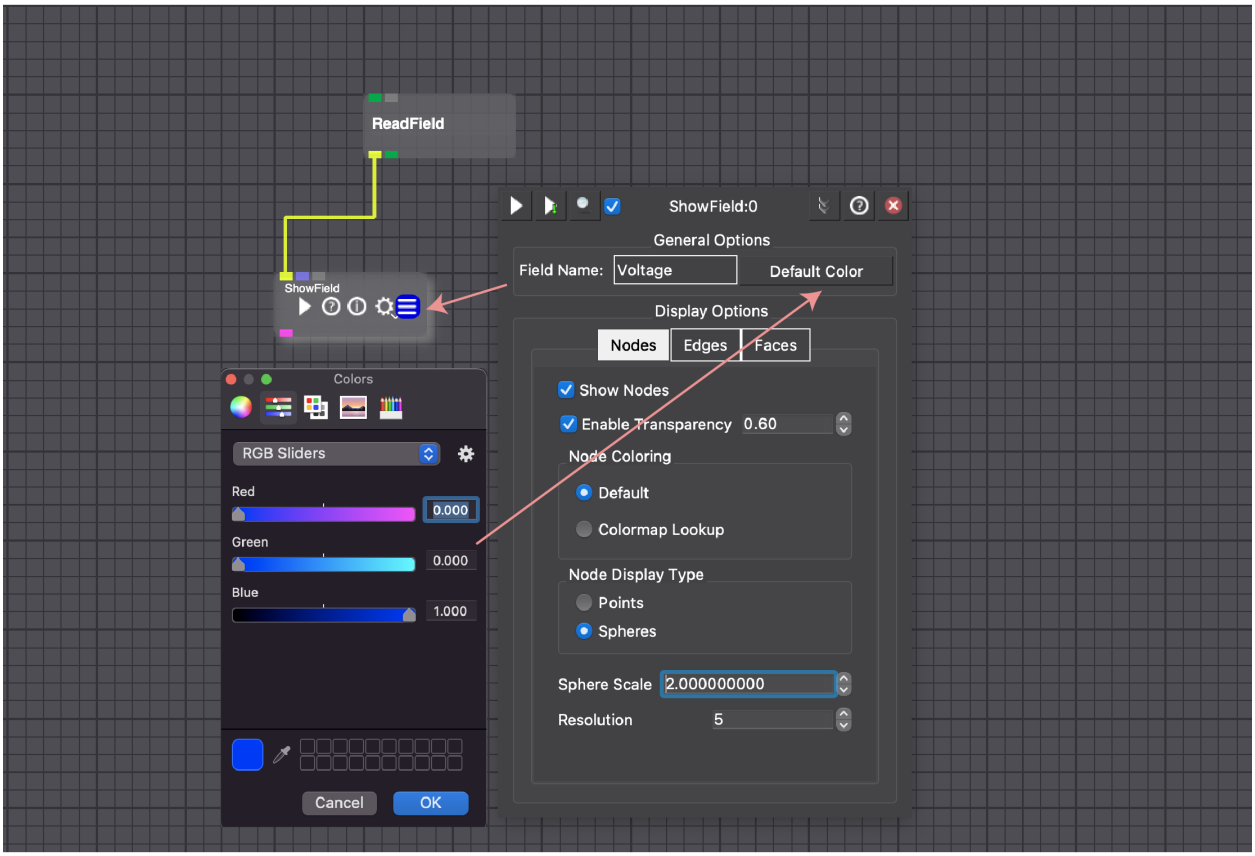


Fig. 3.9: ShowField GUI

Now change the scale and resolution of the nodes.

1. In the **ShowField** GUI, a spin box widget represents the Node Scale. The Node Scale interval can be increased or decreased by a power of 10 by pressing the up and down arrows. Set the Scale to 2. Make sure the Node Display Type is set to Spheres.
2. Set the Sphere Resolution to 5.

3. Go to the Edges tab and turn off the display of edges by deselecting the Show Edges check box.
4. Go the the Faces tab and repeat the same action.
5. Close the ShowField GUI by pressing the *Close* button.

The **ShowField** module is ready to render the nodes as blue spheres. The module Interactively Updates, by default, to execute after every user GUI change. Users can select the Execute button only box to delay all changes until the Execute button is pressed. (This is useful with large dataset, when rendering takes a long time.)

3.6 ViewScene modules

The **ViewScene** is the last module that will be added to the network.

1. Create a **ViewScene** module by selecting the module under the Render section in the module selector.
2. Connect the output port from **ShowField** into the **ViewScene** input port. Notice the **ViewScene** module automatically creates a new input port; this is an example of a SCIRun module that has dynamic input ports. This allows the **ViewScene** module to support an infinite number of geometry producing modules.
3. Open the **ViewScene** window by pressing the **ViewScene**'s UI button.
4. In the **ViewScene** window, there is a set of axes, representing X, Y, and Z directions. To make all of geometry piped to the viewer visible, press the Autoview button located on the top menu of the window. (Note: any time the view is changed (scaled, rotated, or translated), and you want the viewer to re-display everything in the center of the screen, use the Autoview button.) At this point, the utah-torso voltage Field should appear in the **ViewScene** window. It should appear similar to Fig. 3.10, but will be somewhat different as the figure has been scaled and rotated.

3.7 Mouse Controls

In the Viewer, the mouse can be used to rotate, scale, and translate the image.

3.7.1 Translating the image (Right Button)

1. Move the mouse to the center of the image.
2. Click and hold the right mouse button.
3. Move the mouse to translate the image.
4. Release the button, and the image stays in its new location.

3.7.2 Rotating the image (Left Button)

1. Click and hold the left mouse button.
2. Move the mouse to rotate the image.
3. Release the mouse button.

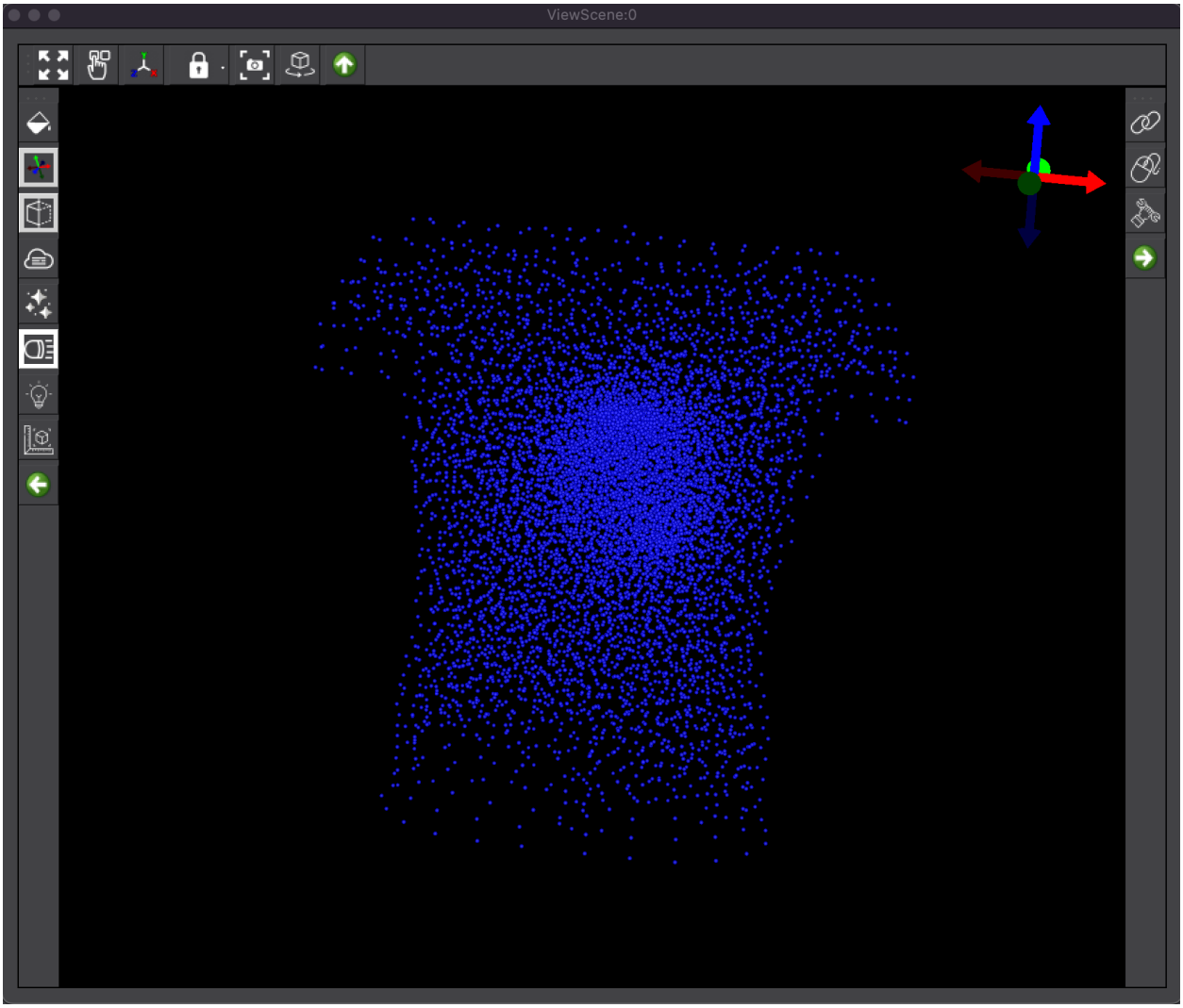


Fig. 3.10: ViewScene window showing the utahtorso-lowres-voltage data

3.7.3 Scaling the scene (Scroll Wheel)

2. Scroll up to zoom the image out.
3. Scroll down to zoom the image in.

3.8 Setting Visualization Parameters

Now review the tools at the top of the ViewScene window (see Fig. 3.11). Buttons at the top of the ViewScene window are used for the following functions (for detailed information about the functionality of each tool, refer to the *ViewScene module documentation*):

- Auto View: restores the display back to a default condition. This is very useful when objects disappear from the view window due to a combination of settings
- Object Selection
- View Options
- Camera Locks
- Screenshot
- Auto Rotate

Buttons on the left side of the ViewScene window (see Fig. 3.11) are used for the following functions:

- Background Color
- Orientation Axes
- Plane Settings
- Fog Controls
- Material Properties
- Headlight
- Additional Lights
- Scale Bar

Buttons on the right side of the ViewScene window (see Fig. 3.11) are used for the following functions:

- Groups
- Mouse Controls
- Report Problems

3.9 Saving and reloading networks

Now that a three-module network has been created, save it to disk. The .srn5 file can easily be reloaded in a future SCIRun session.

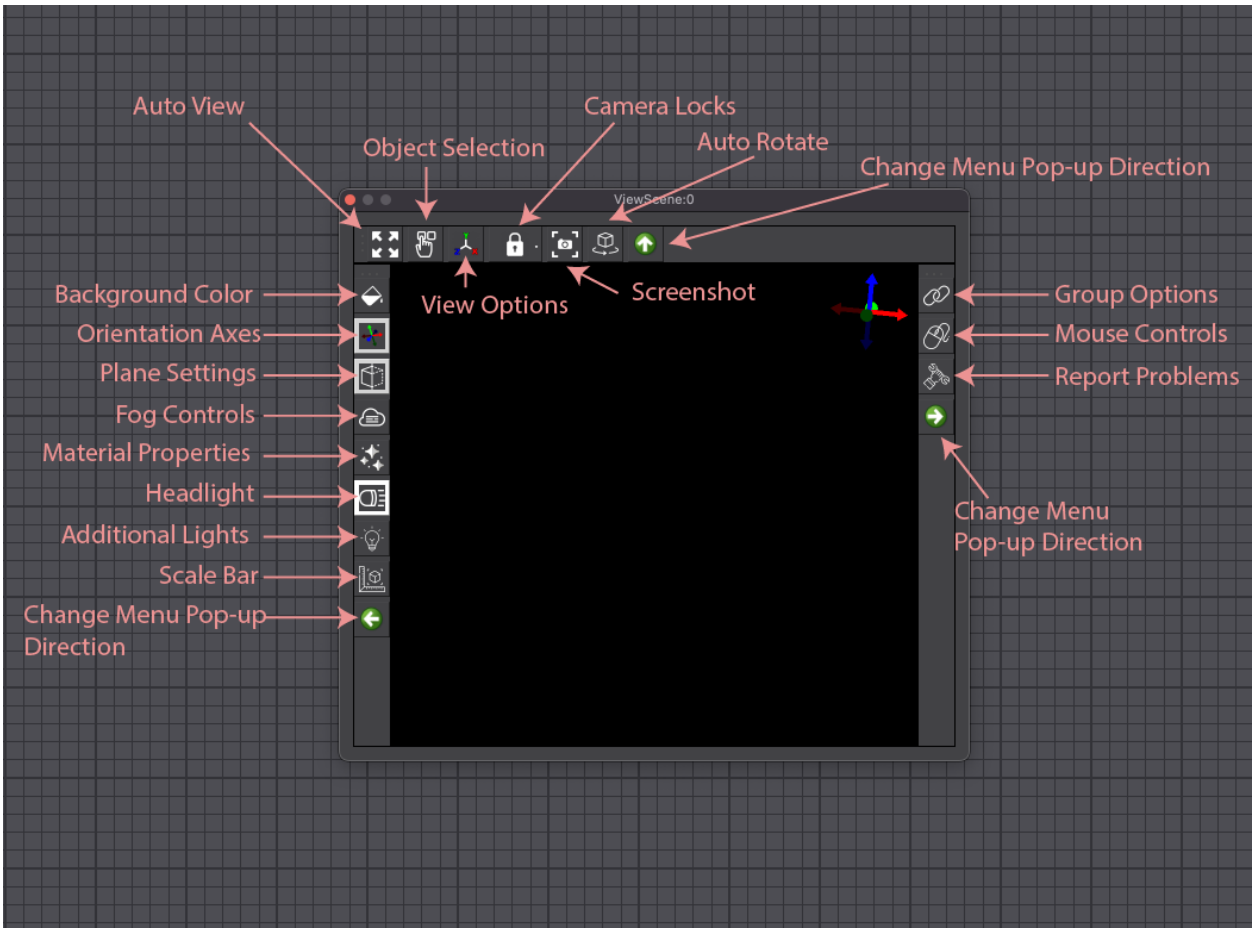


Fig. 3.11: ViewScene window controls

3.9.1 Saving a SCIRun network:

1. Click on the File menu (at the top of the Network Editor window) and select "Save As."
2. When the file browser appears, follow the prompt to choose a location and filename for the network. Many example networks are stored in the ExampleNets directory, which is distributed with the binaries, or in SCIRun/src/ExampleNets. The network can be stored in any location with write access.
3. For this example, store the net as SCIRun/src/nets/show-torso-mesh.srn5, as in Fig. 3.12. The .srn5 suffix is used for SCIRun network files

Please note, to avoid losing work, it is strongly recommended that nets be saved frequently. Auto-save can also be enabled in the SCIRun Preferences window.

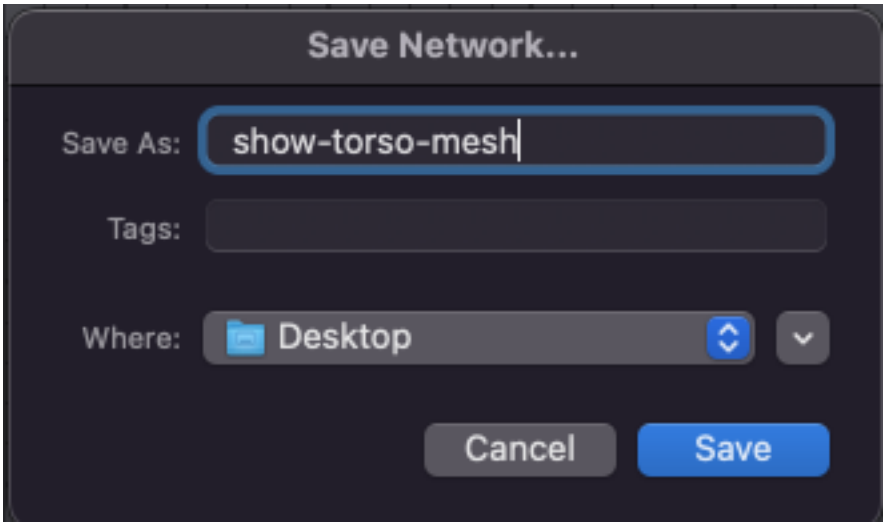


Fig. 3.12: "Save As" GUI

4. Click the Save button. The network is saved, and the dialog disappears.
5. Exit SCIRun by closing the main SCIRun window or pressing Ctrl-Q.

3.9.2 Loading a SCIRun network:

1. Start SCIRun.
2. From the File drop-down menu, select the the Load… option.
3. Select the previously *saved net* (SCIRun/src/nets/show-torso-mesh.srn5).

The net reloads into SCIRun, where it was previously saved. If the net was saved with any of the module UIs open, those UIs automatically re-open when loaded to the net. After changing module settings (e.g., rotating the image in the ViewWindow or changing the rendering color of the nodes in ShowField), there are two options for re-saving the net:

1. Overwrite existing show-torso-mesh.srn5 file by using File->Save from the drop-down menu.
2. Save the net to a new file by using File->Save As…

3.10 Appendix 1

SCIRun has nine geometric meshes available for Fields:

PointCloudMesh: unconnected points

ScanlineMesh: regularly segmented straight line (a regular 1D grid)

CurveMesh: segmented curve

ImageMesh: regular 2D grid (see note below)

Structured Quad Surface mesh: surface made of connected quadrilaterals on a structured grid

Structured Hex Volume mesh: subdivision of space into structured hexagonal elements

TriSurfMesh: surface made of connected triangles

QuadSurfMesh: surface made of connected quadrilaterals

LatVolMesh: regular 3D grid

TetVolMesh: subdivision of space into tetrahedral elements

HexVolMesh: subdivision of space into hexagonal elements

PrismVolMesh: five faces, two triangular faces connected together by three quadrilateral faces.

The following data types can be stored in a Field:

- tensor
- vector
- double precision
- floating point
- integer
- short integer
- char
- unsigned integer
- unsigned short integer
- unsigned char

CHAPTER

FOUR

OLD SCIRUN4 TUTORIALS

CHAPTER

FIVE

PARSER HELP

5.1 Parser Description

5.1.1 Overview

The SCIRun parser is designed to give the user the ability to evaluate almost any expression at the elements of a matrix or field. The underlying engine of the parser assumes that every field or matrix can be represented by an array of scalars, vectors, or tensors. For example a field can be represented by an array of scalars for the field data and an array of vectors for the locations of the nodes. In order to maintain a reasonable performance while interpreting a user defined function, the engine will apply each mathematical operations to the full array at same time. The logic of the parser will allow the user to write several expression at the same time, which will optimized to remove any duplicate function calls and the operations will be untangled in a series of elementary operations that have to be executed.

5.1.2 Using the parser

Each parser has several preloaded variables available to the user. These preloaded variables in SCIRun are always denoted with an upper case variable name. For example for the *CalculateFieldData* has the following variables available:

```
DATA, X,Y,Z, POS
```

The variables can be used in the user to construct a new variable. For example, one can write the following equation:

```
MYDATA = sin(X)+cos(Y);
```

This will assign the value of `sin(X) + cos(Y)` to `MYDATA`, where `X` and `Y` are the arrays containing the `X` and `Y` positions of the nodes. Note that to generate a new variable, no new variable type needs to be declared. The parser will automatically derive the type of the output variable. For example the following case will generate vector data:

```
MYDATA = vector(Y,X,2*Z);
```

As the output of the expression is a vector `MYDATA` will now be an array of vectors instead of an array of scalars as in the previous example.

To assign any data to the output of a module, certain variables need to be assigned. For example in case of the *CalculateFieldData*, the output variable is called `RESULT` and hence a second expression can be added to assign the newly created vector to the output:

```
MYDATA = vector(Y,X,2*Z); RESULT = MYDATA;
```

Now this expression could have been simplified to one expression, which would result in the same operation:

```
RESULT = vector(Y,X,2*Z);
```

From a performance perspective both would be evaluated equally fast as, copying operations are generally removed by the internal optimizer. However to increase user readability one can split an expression in as many pieces as one wants. For example, the following expression is a perfectly valid expression:

```
DX = X-3; DY = Y-4; DZ = Z - 2; RESULT = norm(vector(DX,DY,DZ));
```

Hence the general grammar of the expression is:

`VAR1 = function(...); VAR2 = function(...); VAR3 = function(...);` ... etc

The array parser generally allows two types of input, an array with one element, or an array with many elements. When combining multiple fields or matrices, the arrays with many elements are required to have the same number of elements and the math operations are done by taking corresponding elements from each array. However an array with one element is treated special, and its value is used for every element for the arrays that have many arrays. For example, assume matrix `A = [1]`, and `B = [1,2,3,4]`. We can now evaluate the following expression:

```
RESULT = A+B;
```

In this case `RESULT` will be `[1+1,2+1,3+1,4+1]` and hence allows for scalar parameters to be added into the expressions.

5.1.3 Available functions

The following basic operators have been implemented for Scalars, Vectors, and Tensors:

+, -, *, /, ==, !=, !, &&, ||

In addition to these operators the following functions are available:

- `add()`
- `sub()`
- `mult()`
- `div()`
- `exp()`
- `log()`
- `ln()`
- `log2()`
- `log10()`
- `sin()`
- `cos()`
- `tan()`
- `asin()`
- `acos()`
- `atan()`
- `sinh()`
- `cosh()`
- `asinh()`
- `acosh()`

- pow()
- ceil()
- floor()
- round()
- boolean()
- norm()
- isnan()
- isfinite()
- isinfinite()
- select()
- sign()
- sqrt()
- not()
- inv()
- abs()
- and()
- or()
- eq()
- neq()

For vectors the following functions are available:

To construct a vector use the following function:

```
MYVECTOR = vector(X,Y,Z);
```

To access the components of a vector use the following functions:

```
X = x(MYVECTOR); Y = y(MYVECTOR); Z = z(MYVECTOR);
```

The following vector specific functions are available:

- dot()
- cross()
- normalize()
- find_normal1()
- find_normal2()

For tensors the following functions are available:

To construct a tensor use the following functions:

```
MYTENSOR = tensor(XX,XY,XZ,YY,YZ,ZZ);
```

```
MYTENSOR = tensor(SCALAR);
```

```
MYTENSOR = tensor(EIGVEC1,EIGVEC2,EIGVAl1,EIGVAL2,EIGVAL3);
```

To access the components of a tensor use the following functions:

```
XX = xx(MYTENSOR); XY = xy(MYTENSOR); XZ = xz(MYTENSOR);
YY = yy(MYTENSOR); YZ = yz(MYTENSOR); ZZ = zz(MYTENSOR);
EIGVEC1 = eigvec1(MYTENSOR); EIGVEC2 = eigvec2(MYTENSOR); EIGVEC3 = eigvec3(MYTENSOR)
EIGVAL1 = eigval1(MYTENSOR); EIGVAL2 = eigval2(MYTENSOR); EIGVAL3 = eigval3(MYTENSOR);
```

The following tensor specific functions are available:

`quality()`, `trace`, `det()`, `frobenius()`, `frobenius2()`, `fracanisotropy()`

For converting any Scalar, Vector or Tensor in to a boolean, e.g. whether all components are equal to zero or not, the function `boolean()` provided:

```
RESULT = boolean(vector(X,0,Z));
RESULT = boolean(vector(X,0,Z)) || boolean(tensor(4)*0);
```

The `select()` function works like the C/C++ ternary `?:` operator:

```
RESULT = select(X>2,1,0);
```

To insert a random number two functions are provided, `rand()` and `randv()`. Neither of them take any input. The first one just render a single random number and the second renders an array of the same size as the other arrays. For example to render uniform distributed noise between 0 and 2 for a field, use the following expression:

```
RESULT = 2*randv();
```

If on the other hand all the values need to be the same, but one random value, use :

```
RESULT = rand();
```

Finally, we have an object called element, which refers to the element of a mesh. The latter only exists as an input object, but allows the user to query, properties like element quality metrics and sizes. Currently the following functions are implemented:

`center()`,`volume()`,`size()`,`length()`,`area()`,`dimension()`

CHAPTER

SIX

PYTHON API 0.2

6.1 Global functions

6.1.1 Network editing

- `scirun_add_module("ModuleName")`
  - Adds a new instance of a module to the network. Returns the module ID as a string. (*)
- `scirun_remove_module("ModuleID")`
  - Removes the module specified by the ID string. (*)
- `scirun_execute_all()`
  - Executes the entire current network. (*)
- `scirun_module_ids()`
  - Returns a list of all module ID strings in the current network, sorted by module creation time.
- `scirun_connect_modules("ModuleIDFrom", fromIndex, "ModuleIDTo", toIndex)`
  - Connects ports between two modules by index. From is the output, To is the input. (*)
- `scirun_disconnect_modules("ModuleIDFrom", fromIndex, "ModuleIDTo", toIndex)`
  - Used to disconnect two modules, with the same syntax as connecting. (*)

6.1.2 Network file management

- `scirun_save_network("Filename.srn5")`
  - Saves the current network file. (*)
- `scirun_load_network("Filename.srn5")`
  - Loads the specified network file. (*)
- `scirun_import_network("Filename.srn")`
  - Imports the specified v4 network file. (*)
- `scirun_quit_after_execute()`
  - Enables quitting SCIRun after the next network execution is complete. (*)
- `scirun_force_quit()`
  - Quits SCIRun immediately. (*)

6.1.3 Module state editing

- `scirun_get_module_state("ModuleID", "StateVariableName")`
  - Returns the value of the specified module state variable.
- `scirun_set_module_state("ModuleID", "StateVariableName", value)`
  - Sets the specified module state variable's value.
- `scirun_dump_module_state("ModuleID")`
  - Returns a dictionary with the entire state of the specified module.
- `scirun_get_module_transient_state("ModuleID", "StateVariableName")`
  - Returns the value of the specified module transient state variable.
- `scirun_set_module_transient_state("ModuleID", "StateVariableName", value)`
  - Sets the specified module transient state variable's value. Used to pass data values (strings, matrices, fields [coming soon]) back to modules.

6.1.4 Module/Datatype input

- `scirun_get_module_input_type("ModuleID", portIndex)`
  - Returns the type of the input data object on the specified port.
- `scirun_get_module_input_object("ModuleID", "PortName")`
  - Returns a special `PyDatatype` wrapper object containing a copy of the data on the specified input port, by name.
- `scirun_get_module_input_value("ModuleID", "PortName")`
  - Returns a Python object containing a copy of the data on the specified input port.
- `scirun_get_module_input_object_by_index("ModuleID", portIndex)`
  - Returns a special `PyDatatype` wrapper object containing a copy of the data on the specified input port, by port index.
- `scirun_get_module_input_value_by_index("ModuleID", portIndex)`
  - Returns a Python object containing a copy of the data on the specified input port, by index.

6.1.5 InterfaceWithPython special syntax

- This module lets you set input/output variable names in the module UI. Once this is done (or by using the defaults), one can use assignment syntax to read input data and send output data.
  - Examples
    - `pythonOutput = "hello"` will send a string to the output port associated with `pythonOutput`
    - `field = inputField1` will extract a field object from the input port associated with `inputField1`
    - `outputString1 = "string concat " + inputString1` Input/Output can be combined on the same line.
  - Note: for output variable assignment, make sure to include spaces around the =.

InterfaceWithPython Top-Level Script

In the InterfaceWithPython there is a "Top-Level Script" tab which allows users to run matlab code in a broader scope on execution of the InterfaceWithPython Module. This is helpful if there are variables or modules that are costly to compute or load, yet are used in more than one InterfaceWithPython Module. The *Matlab engine* is one such example. To launch the matlab engine once per session, use the following code in the top-level script:

```
import matlab.engine
print('matlab imported')
eng =  matlab.engine.start_matlab() if (not 'eng' in vars()) else eng
```

This will allow the same matlab engine instance to be called with `eng`. *Note: using the* Matlab code block *does not require this code in the top-level tab.*

6.2 Installing packages

- If there is an installation of the packages with other python builds on the machine, the system path can be used to use those packages. This is likely the easiest way to use python packages in SCIRun, and will also likely to be easiest to maintain. However, to use packages from another Python build, the Python versions will need to match. Once the packages are installed, use the SCIRun triggered events to modify the python path to include these packages.*

6.2.1 Installing Python to maintain Python packages.

The main advantage of doing this step is that a separate Python will likely have pip or multiple packages already installed, unlike the SCIRun Python distribution (we are working on it). Most distributions will likely work; some operating systems come with a python distribution that may work, yet it may be an outdated version. Anaconda is a good choice because most of the commonly used packages will be installed when it is installed. However, the cleanest option will likely be installing Python's own distribution, then to install the packages that you want to use. Both of these distributions should come with pip installed, which makes it easy to install most common packages.

Make sure the python versions of the separate Python installation matches the version that SCIRun is using. To check the Python version in SCIRun, simply open the Python Console, it will print the version upon initialization. You can also get the version when running on the command line by using the `-v` flag.

6.2.2 Installing numpy, scipy, and other major packages with pip

Once you have a separate Python with pip installed, installing packages is usually fairly straightforward. The basic command to install numpy is: `pip install numpy` However, if you have multiple python installations, it may be necessary to specify which pip to use. The pip version will need to match the python version you intend to use. Often the version name is appended to the function call: `pip3.6 install numpy` Yet it may be necessary to point to the specific installation: `/Library/Frameworks/Python.framework/Versions/3.6/bin/pip3.6 install numpy` for instance. If there is an error installing the package, try upgrading pip first: `pip3.6 install --upgrade pip`

6.2.3 Installing Matlab engine for python in SCIRun

Full installation instructions for installing the Matlab engine in Python are found on the mathworks site. Keep in mind that each Matlab version will only support a few Python versions, and the Python version much match what is running in SCIRun (see above). Also, it would probably help if Python Matlab engine installation was in the same location as the other packages to be used in SCIRun. It is helpful to have numpy and scipy installed as well.

To install the Matlab engine in Python run in the terminal:

```
cd "matlab_root"/extern/engines/python
"Python_installation"/bin/python3.6 setup.py build install
```

6.2.4 Adding packages to the Python Path in SCIRun

In order to use the installed packages, SCIRun's Python has to be able to find them. If the package files are located in the Python Path, the can be used with the `import` function. The python path can be modified in many ways, but the most practical for general purpose is to use the triggered events in SCIRun. Triggered events are explained in *another section*, but we will explain how to use it to modify the Python Path.

Open up the Triggered Event Window in SCIRun and choose the "Application Start" tab. Make sure that this script is enabled by clicking the "Enabled" checkbox. Now enter a script that will add the directories to the package installation in the textbox. It will look something like this:

```
import sys
sys.path.append('/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-
↪packages/')
```

There can be many directories or many `sys.path.append(...)` calls as needed. This script is saved and will run every time SCIRun starts, and therefore it will update the path.

If there are module or package paths that need to be set on a network basis (rather than for every network), the InterfaceWithPython module can achieve this with the *"Top-Level Script" Tab*. This tab will execute the code in this tab on a global scale (until SCIRun is closed), so a script similar to the triggered events example will be saved and executed only for the network.

6.3 Matlab engine in SCIRun 5 (through python)

In SCIRun 5, Matlab code and functions can be run using the matlab engine for python in the python console or python interface. To do so, make sure that the matlab engine is installed (previous section). Full documentation can be found here.

6.3.1 Matlab code block

In the InterfaceWithPython module, there is a Matlab code block option. This is a series of Matlab functions that are separated by `%%`. The InterfaceWithPython module will attempt to convert the native Matlab code in the block as Python code. The commands cannot be complex in that they must only contain one function per line. For example:

```
%%
A = magic(3);
[evec, eval] = eig(A);
%%
```

Users can insert a Matlab code block with the button in the InterfaceWithPython UI, or by typing %%. To use the Matlab code block, make sure:

- Matlab and the Python Matlab engine are installed
- The Matlab Python module is in the Python Path in SCIRun
- The "MatlabConversion.py" file is in the Python Path. This is found in the source code: "SCIRun/src/Modules/Python".
- Matlab functions are in the Matlab path. This is best accomplished in the Matlab UI.

The Matlab Code Block is experimental code, so it will likely not work with complex Matlab functionality. However, if there is a code block detected, SCIRun will start Matlab and it will remain open until SCIRun is closed. The python variable names should match the variable names in the assignment.

6.4 Triggered Events

Triggered events in SCIRun execute a python script upon certain SCIRun events. Possible events include: application start, network load, and adding a module. The scripts and settings can be modified in the triggered events window. The triggered event scripts allow for increased customization such as: modifying the Python path, changing the module default settings, and others. Instructions on how to use triggered events to modify the Python path are *described previously*.

To change the default settings of a module, use the 'post module add' trigger. Make sure it is enabled and use `scirun_set_module_state()` function. For example, to change the default matrix type for the ReadMatrix module to matlab type, use the following function:

```
scirun_set_module_state(scirun_module_ids()[-1], 'FileTypeName', 'Matlab Matrix (*.mat)
↪') if scirun_module_ids()[-1].startswith('ReadMatrix') else None
```

As another example, to always open all the ViewScene windows when a network is loaded, use the 'on network load' trigger. With this trigger enabled, use the following command in the script:

```
[scirun_set_module_state(id, '__UI__', True) for id in scirun_module_ids() if id.
↪startswith('ViewScene')]
```

The `__UI__` variable is a hidden state boolean that indicates the open UI window.

6.5 Python Macros

Python Macros in SCIRun are similar to *triggered events* except they execute a python script when macro button is pressed. This feature can be used to do multi-step tasks at the push of a button. As an example, to have a GetMatrixSlice module to play from the beginning, use the following script:

```
# number of the getmatrixslice.   eg, GetMatrixSlice:0
mod_num = 0
scirun_set_module_state("GetMatrixSlice:"+str(mod_num),"SliceIndex",0)
scirun_set_module_transient_state("GetMatrixSlice:"+str(mod_num),"PlayModeActive",1)
scirun_execute_all()
```

The Macro toolbar will need to be visible to access this feature.

6.6 Running Python Scripts

Python scripts can be used for many things in SCIRun, from building networks to batch executing data. To run a script saved to disk within SCIRun, there are a couple options. There is a run script option (looks like a magic wand), which is part of the advanced toolbar. This option will clear any network when a script is run, so it is great for running scripts that build and execute networks. Scripts can also be called as a command line input with the `-s` or `-S` flags. This option is similar to the run script tool, but it allows for passing script arguments after the script filename. For example, in OS X: `/Applications/SCIRun.app/Contents/MacOS/SCIRun -s *script_filename.py* *arg1*`.

To run a script within SCIRun without clearing the network, open and execute the script in the SCIRun python console: `exec(open('*path_to_script/filename.py*').read())`. This syntax can also be used in SCIRun's interactive mode (`-i` flag from the command line).

CHAPTER

SEVEN

SCIRUN MODULE GENERATION

This project was supported by grants from the National Center for Research Resources (5P41RR012553-14) and the National Institute of General Medical Sciences (8 P41 GM103545-14) from the National Institutes of Health.

Authors:
Jess Tate and Garima Chhabra

7.1 SCIRun Overview

This tutorial demonstrates how to create new modules in SCIRun 5.0. It will walk through all the files needed and the basic module structure used by modules. These instructions assume a basic understanding in C++ coding and other basic programming skills

7.1.1 Software requirements

SCIRun 5.0

Download SCIRun version 5.0 from the SCI software portal.

Make sure to update to the most up-to-date version of the source code available, which will include the latest bug fixes.

Alternatively, use Git to clone the SCIRun repository. We suggest creating a fork of the repository so that you can track your changes and create pull requests to the SCIRun repository (*Creating Your SCIRun Fork*).

Compilers, Dependencies Development Tools

SCIRun will need to be built from the source code in order to test and use any modules written. Make sure that Qt, Git, CMake, and the latest C++ compilers for the operating system are installed. More detailed build instructions are available, *Building SCIRun*.

Creating Your SCIRun Fork

With your own GitHub account, go to the SCIRun GitHub page. Click the fork button on the upper right side of the page. It will ask you where to move the fork to, chose your own account. Once the repository is forked, clone it to your local machine with the following command.

```
$git clone https://github.com/[yourgithubaccount]/SCIRun.git
```

After the the code is cloned, navigate to the repository directory and add the upstream path to the original SCIRun repository.

```
$git remote add upstream https://github.com/SCIInstitute/SCIRun.git
```

You should be able to see both your and the original repositories when you use the command:

```
$git remote -v
```

The fork is good to go, but you will need to sync the fork occasionally to keep up with the changes in the main repository. To sync your fork, use the following commands:

```
$git fetch upstream

$git checkout master

$git merge upstream/master
```

You should sync and merge your fork before you start a new module and before you create a pull request.

It is a good practice to create a new branch in your fork for every module you will be adding. The command to create a new branch is:

```
$git checkout -b [branch_name]
```

Please see the GitHub help page for more information.

7.2 Files Needed for a New Module

This section describes the files need to create a module in SCIRun. Each file is described and a template example provided. These template files are all included in the source code in the template directories.

7.2.1 Overview of Files Needed for each Module

There are only three files required to create a module, though more may be needed depending on the function of the module.

In addition to the required module source code and header files *modulename.cc* and *modulename.h*, a module configuration file is needed. The module configuration file *modulename.module* contains a description of the module and its state and names all the files needed for the module to be included in SCIRun.

Simple modules without user interfaces (UIs) can be created with the previously listed three files alone.

However, if the module function needs a UI, there are three additional files needed.

SCIRun can generate a UI for a module without these, but the functionality is limited to nonexistent.

The Qt UI file *modulenameDialog.ui* that describes the graphics and hooks of the UI is created using the Qt UI editor. Module UIs also require source code and header files; *modulenameDialog.cc* and *modulenameDialog.h*.

Most modules, especially those requiring more than minimal code, should also have algorithm code to allow for greater portability and code control. The algorithm code and header files; *modulenameAlgo.cc* and *modulenameAlgo.h*, contain all the computation of the module.

Though it is possible to build modules without algorithm files, it is considered good practice to do so.

It is worth noting that each of the *CMakeLists.txt* files are in the directories of all of the files (except the module config file). See the examples in the following chapters for details.

7.2.2 Module Configuration File

The module configuration file contains all the information needed for the module factory to create necessary linkage and helper files for modules to be properly included into SCIRun. The module configuration file is located in `src/Modules/Factory/Config/`. It is a text file that describes fields specific to the module delimited by curly brackets.

There are three fields: "module", "algorithm", and "UI" and within each field are subfields "name" and "header", and others depending on the field. The following is an example that reflects the template files included in the source code.

```
{
  "module": {
    "name": "@ModuleName@",
    "namespace": "Fields",
    "status": "description of status",
    "description": "description of module",
    "header": "Modules/Template/ModuleTemplate.h"
  },
  "algorithm": {
    "name": "@AlgorithmName@Algo",
    "namespace": "Fields",
    "header": "Core/Algorithms/Template/AlgorithmTemplate.h"
  },
  "UI": {
    "name": "@ModuleName@Dialog",
    "header": "Interface/Modules/Template/ModuleDialog.h"
  }
}
```

This config file example would not build. Specific examples that build and work are found in the following sections of this tutorial (*Simple Module Without UI*, *Simple Module With UI*, *Simple Module With Algorithm*).

As mentioned before, the UI and algorithm files are not required to generate a module, therefore the subfields for the "algorithm" or "UI" fields can be changed to "N/A" to indicate that these files do not exist. Please refer to *Module Config File* section for an example.

7.2.3 Module Source Code

The Module source code consist of a .cc and .h file that code the actual function of the module. However, since most modules use an algorithm file, these files can also be considered as the code that pulls all the relevant information from the algorithm, the UI, and other modules in order to achieve its proper function. These files should be located in the proper directory with the `src/Modules/` directory. For example purposes, we will show and discuss the template files included in the `src/Modules/Template/` directory.

Module Header File

The module header file functions as a typical C++ header file, containing code establishing the module object and structure. The *ModuleTemplate.h* file found in `src/Modules/Template/` provides an example of the kind of coding needed for a module header. The relevant functions are included here, with annotated comments:

```
// makes sure that headers aren't loaded multiple times.
// This requires the string to be unique to this file.
// standard convention incorporates the file path and filename.
 #ifndef MODULES_FIELDS_@ModuleName@_H__
 #define MODULES_FIELDS_@ModuleName@_H__

 #include <Dataflow/Network/Module.h>
 #include <Modules/Fields/share.h>
// share.h must be the last include, or it will not build on windows systems.

namespace SCIRun {
namespace Modules {
namespace Fields {
// this final namespace needs to match the .module file
// in src/Modules/Factory/Config/

  // define module ports.
  // Can have any number of ports (including none), and dynamic ports.
  class SCISHARE @ModuleName@ : public SCIRun::Dataflow::Networks::Module,
    public Has1InputPort<FieldPortTag>,
    public Has1OutputPort<FieldPortTag>
  {
  public:
    // these functions are required for all modules
    @ModuleName@();
    virtual void execute();
    virtual void setStateDefaults();

    //name the ports and datatype.
    INPUT_PORT(0, InputField, Field);
    OUTPUT_PORT(0, OutputField, Field);

    // this is needed for the module factory
    // the arguments of this function could vary as NoAlgoOrUI or ModuleHasUIAndAlgorithm
    MODULE_TRAITS_AND_INFO(NoAlgoOrUI);
  };
}}}
    #endif
```

One of the key functions of this header file is the definition of the ports used by the module. This template uses one input and output, but any number can be used by changing the number and defining all the port types. To use two inputs and outputs:

```
public Has2InputPorts<FieldPortTag,FieldPortTag>,
public Has2OutputPorts<FieldPortTag,FieldPortTag>
```

If no there are no input or output ports, the commands are:

```
public HasNoInputPorts,
public HasNoOutputPorts
```

Dynamic ports are also possible for the inputs.

Dynamic ports are essentially a vector of ports, and are counted as a single port in the header.

For a single dynamic port, then a static port and a dynamic port:

```
public Has1InputPort<DynamicPortTag<FieldPortTag>>,
public Has2InputPorts<FieldPortTag,DynamicPortTag<FieldPortTag>>
```

Here is a list of port tags that can be used in SCIRun:

- MatrixPortTag
- ScalarPortTag
- StringPortTag
- FieldPortTag
- GeometryPortTag
- ColorMapPortTag
- BundlePortTag
- NrrdPortTag
- DatatypePortTag

The header is also where the ports are named and the datatype is declared. It is important for the name of each port to be unique, including all the inputs and outputs. The datatype of each port is specific when the port is declared and named.

This declares the datatype expected by the port and can be a subset of the port tag type, e.g., DenseMatrix instead of Matrix. However, it is better to do this within the module to control the exception.

If there is a UI with the module in question, the state variables are needed to pass data between the module and the UI. State variables are declared here as public (see *Connecting UI to the Module* for an example). The `setStateDefault` function is how the default state variables are set. If there is no UI and therefore no state variables, this function is set to empty in this file (`virtual void setStateDefaults() {};`) and omitted from the .cc file.

Module Code File

The module header file functions as a typical C++ file, containing the functions for the module. The *ModuleTemplate.cc* file found in `src/Modules/Template/` provides an example of the kind of coding needed for a module .cc file. The relevant functions are included here, with annotated comments:

```
#include <Modules/Fields/@ModuleName@.h>
#include <Core/Datatypes/Legacy/Field/Field.h>
#include <Core/Algorithms/Field/@ModuleName@Algo.h>

using namespace SCIRun::Modules::Fields;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Core::Algorithms::Fields;

// this defines the location of the module in the module list.
// "NewField" is the catagory and "SCIRun" is the package.

MODULE_INFO_DEF(@ModuleName@, NewField, SCIRun) ;


@ModuleName@::@ModuleName@() : Module(staticInfo_)
{
  //initialize all ports.
  INITIALIZE_PORT(InputField);
  INITIALIZE_PORT(OutputField);
}

void @ModuleName@::setStateDefaults()
{
  auto state = get_state();
  setStateBoolFromAlgo(Parameters::Knob1);
  setStateDoubleFromAlgo(Parameters::Knob2);
}

void @ModuleName@::execute()
{
  // get input from ports
  auto field = getRequiredInput(InputField);
  // get parameters from UI
  setAlgoBoolFromState(Parameters::Knob1);
  setAlgoDoubleFromState(Parameters::Knob2);
  // run algorithm code.
  auto output = algo().run(withInputData((InputField, field)));
  //send to output port
  sendOutputFromAlgorithm(OutputField, output);
}
```

As shown in this template example, the module.cc file contains mostly constructors and sends the inputs to the SCIRun algorithm. Most modules should follow this practice, which allows for easier maintenance of common algorithms.

7.2.4 Module UI Code

Three files are needed to set up a UI for a module; a design file, a header file, and a .cc file. These files are located in the same directory within `src/Interface/Modules/`.

We show the examples located in `src/Interface/Modules/Template` as examples of the core functions needed.

Module Design File

The module design file is an xml file that describes the UI structure. This file is created and edited in the Qt editor. The image below shows the example template *ModuleDesignerFile.ui* within the Qt editor.

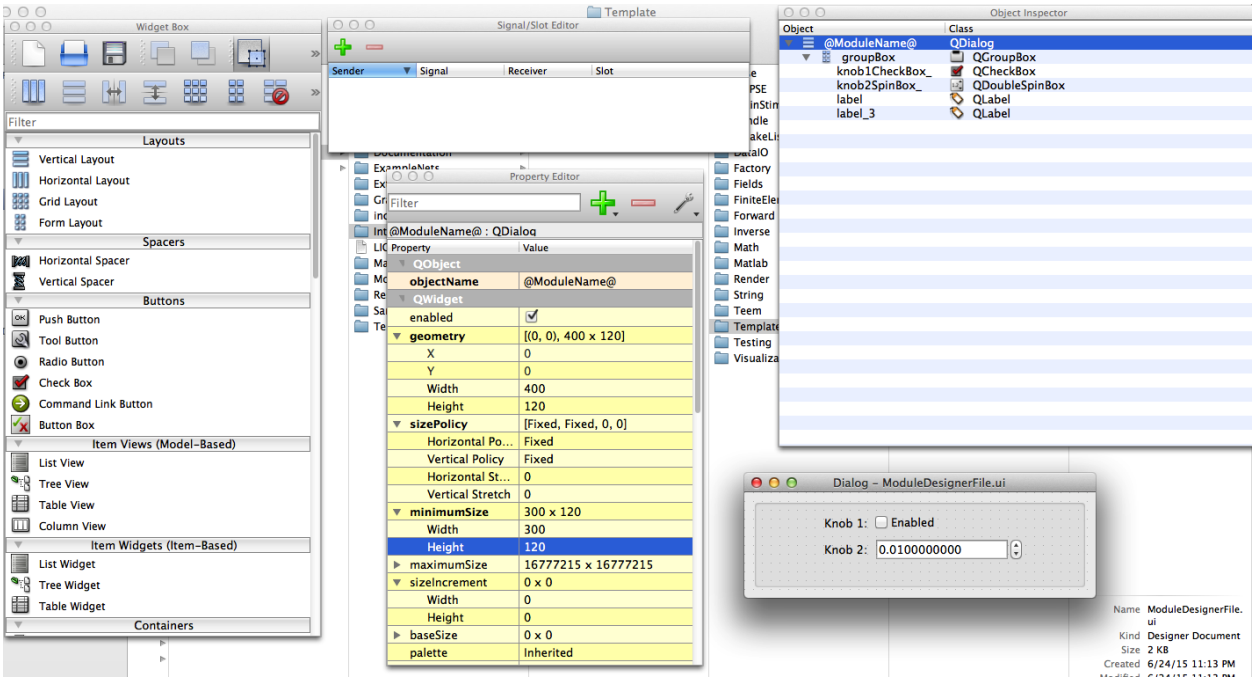


Fig. 7.1: Template module interface design file as seen in the Qt editor

As shown, the user interactively modifies the placement of widgets in the window. The Widget Box window allows the user to choose and place new objects within the window. The Property Editor allows modification of properties of the various objects and widgets within the UI, including size, type of input, names, etc. The hierarchy and organization of the UI can be changed within the Object Inspector window .

When using the editor to make a module UI, there are a few things to consider. First, make sure all the relevant objects, including the name of UI (QDialog) is consistent with module dialog code. You can change the size and placement of objects with the property manager, but make sure to leave some buffer space as some operating systems interpret the file slightly differently. The structure of the UI can be changed or destroyed.

Look at some of the existing modules for examples.

Module Dialog Header

The module dialog header performs as a traditional C++ header for the module dialog code. Shown here is the example *ModuleDialog.h* in the `src/Interface/Modules/Template` folder.

```
 #ifndef INTERFACE_MODULES_@ModuleName@DIALOG_H
 #define INTERFACE_MODULES_@ModuleName@DIALOG_H

//This file is created from the @ModuleName@Dialog.ui in the module factory.
 #include <Interface/Modules/Fields/ui_@ModuleName@Dialog.h>
 #include <boost/shared_ptr.hpp>
 #include <Interface/Modules/Base/ModuleDialogGeneric.h>
 #include <Interface/Modules/Fields/share.h>

namespace SCIRun {
namespace Gui {

class SCISHARE @ModuleName@Dialog : public ModuleDialogGeneric,
  public Ui::@ModuleName@
{
    Q_OBJECT

public:
  @ModuleName@Dialog(const std::string& name,
    SCIRun::Dataflow::Networks::ModuleStateHandle state,
    QWidget* parent = nullptr);
    //this function would be from pulling data from module,
    // usually to change the UI.
  void pull() override;
};
}}
 #endif
```

The module dialog header file is similar for each module, with only the names of the module and a few different functions declared.

Module Dialog Code

The module dialog .cc file is used with Qt to establish the functionality of a module UI.

Shown here is the example *ModuleDialog.cc* in the `src/Interface/Modules/Template` folder.

```
#include <Interface/Modules/Fields/@ModuleName@Dialog.h>
#include <Core/Algorithms/Field/@ModuleName@Algo.h>

using namespace SCIRun::Gui;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Core::Algorithms::Fields;

@ModuleName@Dialog::@ModuleName@Dialog(const std::string& name,
  ModuleStateHandle state,
  QWidget* parent /* = nullptr */)
  : ModuleDialogGeneric(state, parent)
```

(continues on next page)

(continued from previous page)

```
{
  setupUi(this);
  setWindowTitle(QString::fromStdString(name));
  fixSize();

//get values from UI and send to algorithm
  addCheckBoxManager(knob1CheckBox_, Parameters::Knob1);
  addDoubleSpinBoxManager(knob2SpinBox_, Parameters::Knob2);
}

void @ModuleName@Dialog::pull()
{
// pull the code from the module and set in the dialog.
// make changes necessary.
  pull_newVersionToReplaceOld();
}
```

The module dialog code is used mostly for passing data between the module and the UI and changing the UI when required by the module. Parameters and inputs are passed straight to the algorithm, as shown in this example. The 'pull' functions are optional, and use data from the module and algorithm to either display in the UI or to change the options or appearance of the UI. There are other forms of 'pull', such as `pullSpecial`. A `push` function can be used in conjunction with `pull`.

These functions, `pull` and `push` are automatic functions in the module UI.

7.2.5 Algorithm Code

The Module algorithm files are where most of the computation code should live. There are two files, a header and a .cc file. Making algorithm files can be tricky for a beginner, because there are several options to use due to the flexible nature of SCIRun. However, the module algorithms have more in common than initially apparent.

The trick to implementing a new module algorithm (as with most things) is to look at several other modules that have similar functions and try to emulate those methods.

Module algorithm code belongs in the relevant directory within the `src/Core/Algorithms/` directory.

Some template examples are provided for this section, and are found in `src/Core/Algorithms/Template/`

Module Algorithm Header

The module algorithm header performs as a traditional C++ header for the module algorithm code. The example *AlgorithmTemplate.h* found in the `src/Core/Algorithm/Template` folder is shown below.

```
#ifndef CORE_ALGORITHMS_FIELDS_@AlgorithmName@_H
#define CORE_ALGORITHMS_FIELDS_@AlgorithmName@_H

#include <Core/Algorithms/Base/AlgorithmBase.h>
#include <Core/Algorithms/Field/share.h>

namespace SCIRun {
  namespace Core {
    namespace Algorithms {
      namespace Fields {
```

(continues on next page)

(continued from previous page)

```
// declare parameters and options in header when not part of standard names.
        ALGORITHM_PARAMETER_DECL(Knob1);
        ALGORITHM_PARAMETER_DECL(Knob2);

        class SCISHARE @AlgorithmName@Algo : public AlgorithmBase
        {
        public:
          @AlgorithmName@Algo();
          virtual AlgorithmOutput run(const AlgorithmInput& input) const;
        };
      }}}}
#endif
```

A key difference in the algorithm header file are the function and variable declarations.

Parameters and options for the algorithm are declared here if they are not included in the recognized list (listed in `Core/Algorithms/Base/AlgorithmVariableNames.h`).

Module Algorithm Code

The module algorithm .cc file contains the majority of the computation necessary for the module.

The example *AlgorithmTemplate.cc* found in the `src/Core/Algorithm/Template/` folder is shown below

```
 #include <Core/Algorithms/Field/@AlgorithmName@Algo.h>
 #include <Core/Algorithms/Base/AlgorithmVariableNames.h>
 #include <Core/Algorithms/Base/AlgorithmPreconditions.h>
 #include <Core/Datatypes/Legacy/Field/FieldInformation.h>
 #include <Core/Datatypes/Legacy/Field/VField.h>
 #include <Core/Datatypes/Legacy/Field/VMesh.h>
 #include <Core/Logging/Log.h>

using namespace SCIRun;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Core::Algorithms;
using namespace SCIRun::Core::Algorithms::Fields;

//  this function is for setting defaults for state variables.
//  Mostly for UI variables.
@AlgorithmName@Algo::@AlgorithmName@Algo()
{
  using namespace Parameters;
  addParameter(Knob1, false);
  addParameter(Knob2, 1.0);
}

//main algorithm function
AlgorithmOutput @AlgorithmName@Algo::run(const
     AlgorithmInput& input) const
{
  auto inputField = input.get<Field>(Variables::InputField);
```

(continues on next page)

(continued from previous page)

```
  FieldHandle outputField(inputField->deep_clone());
  double knob2 = get(Parameters::Knob2).toDouble();
  if (get(Parameters::Knob1).getBool())
  {
    // do something
  }
  else
  {
    // do something else
  }

  AlgorithmOutput output;
  output[Variables::OutputField] = outputField;
  return output;
}
```

This template uses the Variable namespace, which provides defined names from`Core/Algorithms/Base/AlgorithmVariableNames.h`. This allows for easy use of common input and output to the algorithm. If other or more values are needed, they are declared in the header. Also of note, the default values for the UI are set in the `@AlgorithmName@Algo()` function, and then in the module code, the `setStateDefault` function pulls the values from the algorithm.

This only works if the algorithm files are linked in the module configuration file.

There are several algorithms already implemented in SCIRun. If there are modules that have similar functionality you may be able to use some of the functionality already implemented. The module may still need it's own algorithm file.

7.3 Example: Simple Module Without UI

This section describes how to create a very simple module in SCIRun. We will show how to make a simple module that outputs a simple string. This example shows the basics of the functions and code used by SCIRun to create and run modules.

7.3.1 Module Config File

If you have created a fork from the SCIRun Git repository begin by creating a new branch in your repository. Be sure to commit your changes to your repository often as this can help you and the developers fix and make improvements to the code. It is often easiest to modify existing code to fit your purposes than create new code, so determine a module that has similar functionality or structure to the new module. If desired, there are also several template files described in *Files Needed for a New Module* to use as a basis. In this example, we provide the code needed, so it is not necessary to copy another module.

Begin with the module config file. Create a new text file in the module factory configuration directory (`src/Modules/Factory/Config/`) for the new module. Name it *TestModuleSimple.module* or something similar. The text of the file is:

```
{
  "module": {
    "name": "TestModuleSimple",
    "namespace": "StringManip",
    "status": "new module",
```

(continues on next page)

(continued from previous page)

```
    "description": "This is a simple module to show how to make new modules.",
    "header": "Modules/String/TestModuleSimple.h"
  },
  "algorithm": {
    "name": "N/A",
    "namespace": "N/A",
    "header": "N/A"
  },
  "UI": {
    "name": "N/A",
    "header": "N/A"
  }
}
```

The exact text of the status and description is whatever the developer desires. The names of the module and filenames can be different, but they must match the module code.

7.3.2 Module Header File

Now we move on to the module code. The module is placed in one of the directories in `src/Modules/`, so choose the directory that fits the modules use best (do not place the module code in *Factory* or *Template*, and *Legacy* is generally for converted modules from earlier versions of SCIRun). Since this module has a simple string output, we place the module code in `src/Modules/String/`. Create a file called *TestModuleSimple.h* in this directory. This file is similar to the *ModuleTemplate.h* file shown earlier. In addition to the SCIRun license information, the content of the header file is:

```
#ifndef MODULES_STRING_TestModuleSimple_H
#define MODULES_STRING_TestModuleSimple_H

#include <Dataflow/Network/Module.h>
#include <Modules/Fields/share.h>

namespace SCIRun {
namespace Modules {
namespace StringManip {

class SCISHARE TestModuleSimple : public SCIRun::Dataflow::Networks::Module,
public HasNoInputPorts,
public Has1OutputPort<StringPortTag>
{
public:
  TestModuleSimple();
  virtual void execute();
  virtual void setStateDefaults() {};

  OUTPUT_PORT(0, OutputString, String);

  MODULE_TRAITS_AND_INFO(SCIRun::Modules::ModuleFlags::NoAlgoOrUI);
};
}}}
#endif
```

As mentioned in *Module Configuration File*, the header files for most modules do not vary significantly. This example in particular contains only elements common to most other modules. The key to creating the header files is to ensure that the module name is correct in every place it occurs, that the namespace (StringManip) matches the module config file and that the ports are numbered and labeled correctly.

If desired, the final version of the header file is in the source code: `src/Modules/Examples/TestModuleSimple.h`.

7.3.3 Module Source Code

The final file needed for this module is the source code file; TestModuleSimple.cc.

The functionality used in this module is minimal to show essential functions. With the license and other comments, the file should contain:

```
#include <Modules/String/TestModuleSimple.h>
#include <Core/Datatypes/String.h>

using namespace SCIRun;
using namespace SCIRun::Modules::StringManip;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Dataflow::Networks;

/// @class TestModuleSimple
/// @brief This module splits out a string.

const ModuleLookupInfo TestModuleSimple::staticInfo_("TestModuleSimple","String","SCIRun
↪");

TestModuleSimple::TestModuleSimple() : Module(staticInfo_,false)
{
  INITIALIZE_PORT(OutputString);
}

void
TestModuleSimple::execute()
{
  std::string message_string;

  message_string = "[Personalize your  message here.]";

  StringHandle msH(new String(message_string));
  sendOutput(OutputString, msH);
}
```

7.3.4 Building and Testing

For comparison, the final version of the module code file is in `src/Modules/Examples/TestModuleSimple.cc`.

After these files are modified correctly, the only step remaining before building is adding the module code and header to the compiler list.

Open the `src/Modules/String/CMakeLists.txt` file.

Add *TestModuleSimple.cc* and *TestModuleSimple.h* to the respective list. There is more to the file, but the relevant sections should look something like this:

```
SET(Modules_String_SRCS
  CreateString.cc
  NetworkNotes.cc
  TestModuleSimple.cc
)

SET(Modules_String_HEADERS
  CreateString.h
  NetworkNotes.h
  share.h
  TestModuleSimple.h
)
```

After changing the CMakeList.txt file, build SCIRun using the build script, or if you have already built SCIRun recently, go to the `SCIRun_root/bin/SCIRun` directory and run make.

Take note of any build errors, if there is a problem with any module factory files, make sure that there are no mistakes in the the module configuration file and build again.

Check out the common build errors in *Common Build Errors*.

After SCIRun builds completely, Launch SCIRun and test the module. You can use the PrintDatatype module to view the string that this module outputs by running the 2 module network and then opening the PrintDatatype User Interface. Other modules require more testing, but due to the simple nature of this module you can know that if the message matches what you expect, then it is working properly.

7.4 Example: Simple Module With UI

In this chapter, we build off the module that we described in the previous chapter to show how to add a UI and an input port. This module prints a message that comes from either the input port or the UI. We show how to add a UI incrementally to help convey the principles that the software is based upon. This incremental approach allows the user to copy this approach with more complicated modules as it provides sanity checks for the user.

7.4.1 Duplicate the Previous Module

To begin, copy the *TestModuleSimple.module* in the `src/Modules/Factory/Config/` and name the copy *TestModuleSimpleUI.module*. Change the name and header field to reflect the new name of the module, as shown here:

```
  "module": {
    "name": "TestModuleSimpleUI",
    "namespace": "StringManip",
    "status": "new module",
    "description": "This is a simple module to show how to make new modules.",
    "header": "Modules/String/TestModuleSimpleUI.h"
  },
```

For now, leave the rest of the fields as 'N/A'; we will come back to those.

Next, copy the module code files *TestModuleSimple.h* and *TestModuleSimple.cc* in the `src/Modules/String/` directory and rename them appropriately (*TestModuleSimpleUI.h* and *TestModuleSimpleUI.cc*).

In these new files, change all the references of the module's name to TestModuleSimpleUI.

A find and replace function will manage most instances, but make sure that all of them are changed.

There are 4 lines in each of the two files that need to be changed, with more than one change in some lines. The changes in *TestModuleSimpleUI.h* are shown below:

```
#ifndef MODULES_STRING_TestModuleSimpleUI_H
#define MODULES_STRING_TestModuleSimpleUI_H

...

class SCISHARE TestModuleSimpleUI : public SCIRun::Dataflow::Networks::Module,

...

public:
  TestModuleSimpleUI();

...
```

For the *TestModuleSimpleUI.cc* file:

```
#include <Modules/String/TestModuleSimpleUI.h>


const ModuleLookupInfo TestModuleSimpleUI::staticInfo_("TestModuleSimpleUI",
       "String", "SCIRun");

TestModuleSimpleUI::TestModuleSimpleUI() : Module(staticInfo_)


void
TestModuleSimpleUI::execute()
{
```

Another change you may notice is to remove the `false` input in the constructor line:

```
TestModuleSimpleUI::TestModuleSimpleUI() : Module(staticInfo_)
```

The *'false'* option means that there is no module UI.

Removing the option changes the input to 'true', which allows for a module UI.

If no UI file is found, a default UI is used.

With these changes we should try to build. Make sure the files are added to the CMakeList.txt file in `src/Modules/String/` as shown in the previous example. If there are build errors, check for spelling mismatches.

Also, check out the common build errors in *Common Build Errors*. Once SCIRun is built, you can try to add the new module to the workspace. SCIRun will give you a warning dialogue about not finding a UI file, so it will create a default one.

This UI is not connected to anything, so it won't affect the module at all, but you should be able to open the UI and see it (a slider and two buttons).

Check to make sure that the output is still the string that you expected.

If everything is working properly, we can move onto the next step of adding our own module.

7.4.2 Creating a Custom UI

To create a new UI, we add three new files: a design file, a UI source code file and a UI header file. These files are linked to the other module code. To do that we modify the module config file again to add the name of the UI and the path to the header file. The naming convention often used is to add 'Dialog' to the end of the module name for the name of the UI and the names of the files.

```
"UI": {
    "name": "TestModuleSimpleUIDialog",
    "header": "Interface/Modules/String/TestModuleSimpleUIDialog.h"
  }
```

Next, we use the Qt editor to design a module UI. Copy the Qt module file from `src/Interface/Modules/Template/ModuleDesignerFile.ui` to `src/Interface/Modules/String/TestModuleSimpleUIDialog.ui`.

Open the *TestModuleSimpleUIDialog.ui* file in the Qt editor, which provides a graphic method for modifying and compiling the design file. First, delete the check box, input scroll wheel widgets, and the `knob 2` label. Next, add a `line edit widget` by finding it in the *Widgdet Box* window (in the *Input Widget* section) then clicking and dragging it next to the remaining label.

Change the text of the remaining label to `My String:` or something similar.

Finally, change the name of the of the `line edit widget` to `inputstring_` and the name of the `QDialog| object` to *TestModuleSimpleUIDialog*.

This can be done in the *Object Inspector* or in the *Property Editor* when the appropriate object is clicked.

Fig. 7.2 shows what the module should look like in the Qt editor.

Now that the module UI is designed, we need to link it to the module with the module dialog code.

Copy the *ModuleDialog.cc* and the *ModuleDialog.h* from the `src/Interface/Modules/Template/` directory to the `src/Interface/Modules/String/` directory, with the appropriate names (*TestModuleSimpleUIDialog.cc* and *TestModuleSimpleUIDialog.h*). For the *TestModuleSimpleUIDialog.h*, change the module name reference to the correct module name and delete the 'virtual void pull()' function. The code should be very similar to the following:

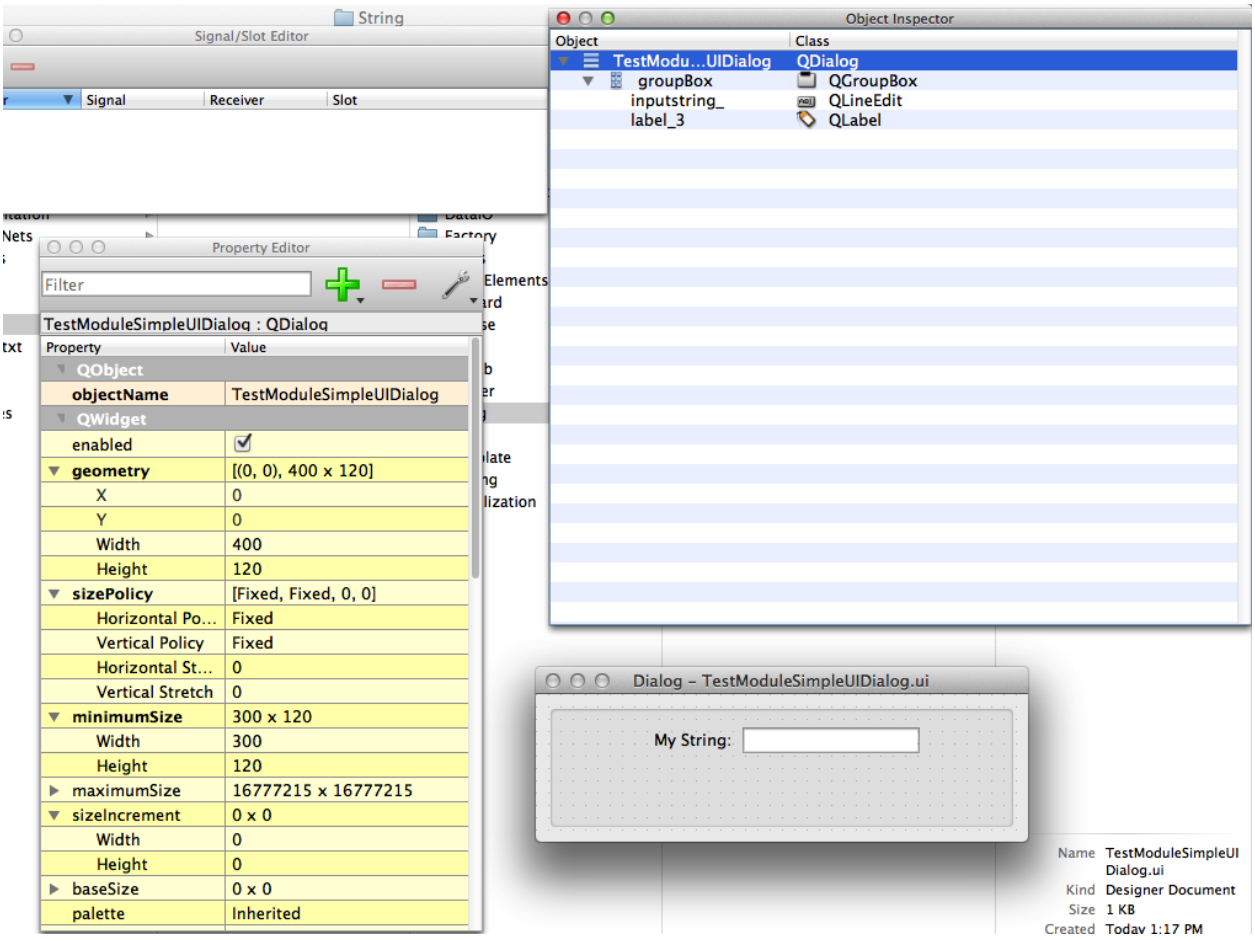


Fig. 7.2: Module interface design file for the TestModuleSimpleUI module as seen in the Qt editor.

```
#ifndef INTERFACE_MODULES_STRING_TestModuleSimpleUIDialog_H
#define INTERFACE_MODULES_STRING_TestModuleSimpleUIDialog_H

#include <Interface/Modules/String/ui_TestModuleSimpleUIDialog.h>
#include <Interface/Modules/Base/ModuleDialogGeneric.h>
#include <Interface/Modules/String/share.h>

namespace SCIRun {
namespace Gui {

class SCISHARE TestModuleSimpleUIDialog : public ModuleDialogGeneric,
  public Ui::TestModuleSimpleUIDialog
{
    Q_OBJECT

public:
  TestModuleSimpleUIDialog(const std::string& name,
    SCIRun::Dataflow::Networks::ModuleStateHandle state,
    QWidget* parent = nullptr);
};
}}
#endif
```

The *TestModuleSimpleUIDialog.cc* requires similar treatment, but requires the addition of a few more changes. Add an include for the module header file, change the namespace from `Field` to `StringManip`, delete the last two lines from the main function, and delete the small function found after the main function.

The code should be:

```
#include <Interface/Modules/String/TestModuleSimpleUIDialog.h>
#include <Modules/String/TestModuleSimpleUI.h>

using namespace SCIRun::Gui;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Modules::StringManip;

TestModuleSimpleUIDialog::TestModuleSimpleUIDialog(const std::string& name,
  ModuleStateHandle state,
  QWidget* parent /* = nullptr */)
  : ModuleDialogGeneric(state, parent)
{
  setupUi(this);
  setWindowTitle(QString::fromStdString(name));
  fixSize();
}
```

This is enough to create a UI for the TestModuleSimpleUI module, but it will not be able to interact yet. We modify this file later to connect all the required inputs. For now, build SCIRun to test the UI design. Make sure that these three new files are added to the *CMakeList.txt* in the `src/Interface/Modules/String/` directory:

```
SET(Interface_Modules_String_FORMS

  ...
```

(continues on next page)

(continued from previous page)

```
  TestModuleSimpleUIDialog.ui
)

SET(Interface_Modules_String_HEADERS



  TestModuleSimpleUIDialog.h
)

SET(Interface_Modules_String_SOURCES

  ...

  TestModuleSimpleUIDialog.cc
)
```

Once these files are added, SCIRun should build. Load SCIRun and place the TestModuleSimpleUI module. Open the UI for the module and make sure that it looks correct.

7.4.3 Connecting UI to the module

We now work on connecting the input from the UI to the code in the module. Begin by modifying the *TestModuleSimpleUIDialog.cc* to include a line that reads the input field and assigns it to a variable.

This line is placed near the end of the main function in the module dialog code

```
addLineEditManager(inputstring_,TestModuleSimpleUI::FormatString);
```

This function reads the value of `inputstring` and sets it to `FormatString`, which we have included as if it was part of the `TestModuleSimpleUI` namespace.

We include it as such, by adding it as a public function in the *TestModuleSimpleUI.h* file.

```
static Core::Algorithms::AlgorithmParameterName FormatString;
```

This is the final declaration in the public list (after `MODULE_TRAITS_AND_INFO(ModuleHasUI)_`). Another change in this file is to modify the `setStateDefault` function so that it is not empty. Remove the curly brackets from this:

```
virtual void setStateDefaults() {};
```

so that it is:

```
virtual void setStateDefaults();
```

We need a couple more additions to make the value from the UI available for use in the main function code. In the *TestModuleSimpleUI.cc* file, add the following line before the main execute function, i.e., right after declaring the namespaces.

```
SCIRun::Core::Algorithms::AlgorithmParameterName
TestModuleSimpleUI::FormatString("FormatString");
```

Next, we set the state defaults by creating context for the 'setStateDefault' function we just exposed. Add this function just before the execute function.

```
void TestModuleSimpleUI::setStateDefaults()
{
  auto state = get_state();
  state->setValue(FormatString,std::string ("[Insert message here]"));
}
```

With these three additions, the code should build.

If you load the module, you should see the default message (`"[Insert message here]"`) in the input field. Changing this will still not affect the output because the execute function is still hard coded for a specific message.

We now change the execute function to use the UI inputs. Simply get the state of the module (`auto state = get_state();`), then assign the output string variable to `state -> getValue(FormatString).toString();. The function is as follows:

```
void
TestModuleSimpleUI::execute()
{
  std::string message_string;
  auto state = get_state();
  message_string = state -> getValue(FormatString).toString();
  StringHandle msH(new String(message_string));
  sendOutput(OutputString, msH);
}
```

After building the software, you should now see that the output of module will be the same as the string that is put in the input field in the module UI.

7.4.4 Adding an Input Port

With the UI implemented and working, we now add an optional input port to the module. This functionality is simple in SCIRun 5. We add the port in the *TestModuleSimpleUI.h* file. First, replace the line:

```
 public HasNoInputPorts,
```

with:

```
public Has1InputPort<StringPortTag>,
```

Next, we name the port in the list of public objects. Add:

```
INPUT_PORT(0, InputString, String);
```

near the output port declaration. These changes are all that are needed in the header file, but we need to initialize the port in the .cc file. In the *TestModuleSimpleUI.cc*, add the initializing line to the module constructor function:

```
TestModuleSimpleUI::TestModuleSimpleUI() : Module(staticInfo_)
{
  INITIALIZE_PORT(InputString);
  INITIALIZE_PORT(OutputString);
}
```

These changes allow the code to build with an input port, yet the input will not yet affect the output of the module.

In the main execute function in *TestModuleSimpleUI.cc*, we read whether there is an input, then use that input or the UI input if there is none. The execute function is:

```
void
TestModuleSimpleUI::execute()
{
  std::string message_string;
  auto  stringH = getOptionalInput(InputString);
  auto state = get_state();

  if (stringH && *stringH)
  {
    state -> setValue(FormatString, (*stringH) -> value());
  }

  message_string = state -> getValue(FormatString).toString();
  StringHandle msH(new String(message_string));
  sendOutput(OutputString, msH);
}
```

This code reads an optional input, checks if it is not empty, and if so then changes the state variable to the input. By changing the state variable before assigning it to the output, it changes the UI input string also.

This is all the changes necessary to add inputs to this module. Build SCIRun, then test the module using the CreateString and PrintDatatype modules. When there is no input, the value in the UI field is the output. When there is an output, the input port is the same as the output port, and the UI input field is set to the input string. This prevents the user from changing the input string while there is a string in the input port.

For a slightly more complicated, yet much more useful module as an example, check out PrintStringIntoString. The setup code is mostly the same, except there are dynamic ports, so much of the code looks similar.

7.4.5 Finished Code

For the sake of comparison, the final version of the code for this module is included in the source code in the Example files. The module code files are in `src/Modules/Examples/`, *TestModuleSimpleUI.cc* and *TestModuleSimpleUI.h*. The module UI code files are in `src/interface/Modules/Examples/`, *TestModuleSimpleUIDialog.cc*, *TestModuleSimpleUIDialog.h*, and *TestModuleSimpleUIDialog.ui*.

7.5 Example: Simple Module With Algorithm

In this chapter, we show how to build a module with a simple algorithm and a simple UI. This chapter builds off the principles established in the previous examples. We use SCIRun to create a module that performs a simple sorting algorithm on a matrix. This example shows how to use module algorithm files with a module UI to implement simple algorithms into modules.

7.5.1 Module Overview

As mentioned in the chapter introduction, we create a module that sorts the entries of a matrix in ascending or descending order. We call the module SortMatrix. This module uses a simple simple quicksort algorithm with a Lomuto partition scheme. There are some implementations for vector sorting in the STL algorithm library, but this implementation works more generally on matrices and will hopefully be helpful in showing how to implement an algorithm from scratch.

There are eight files needed in total for this module: a module configuration file, module code and header file, a UI design file with UI code and header files, and algorithm code and header files. The first six were used in the previous example, but this chapter shows how to incorporate the algorithm code and how it interacts with the module and UI code. Each file is also described in general in *Files Needed for a New Module* with templates.

7.5.2 Module Configuration File

As with the other examples, we need a module configuration file for this module. This file need every field filled. Create a *SortMatrix.module* file in the `src/Modules/Factory/Config/` directory containing the following:

```
{
  "module": {
    "name": "SortMatrix",
    "namespace": "Math",
    "status": "new module.",
    "description": "sorts a matrix.",
    "header": "Modules/Math/SortMatrix.h"
  },
  "algorithm": {
    "name": "SortMatrixAlgo",
    "namespace": "Math",
    "header": "Core/Algorithms/Math/SortMatrixAlgo.h"
  },
  "UI": {
    "name": "SortMatrixDialog",
    "header": "Interface/Modules/Math/SortMatrixDialog.h"
  }
}
```

If you copy another module config file, make sure all the names are correct and that the namespace is set to Math.

7.5.3 Module Code

The next files needed for this module are the module code (*SortMatrix.cc*) and the header (*SortMatrix.h*) files. These files are located in `src/Modules/Math/`.

The header (*SortMatrix.h*) file is not much different from the other two examples, as shown here:

```
#ifndef MODULES_MATH_SortMatrix_H
#define MODULES_MATH_SortMatrix_H

#include <Dataflow/Network/Module.h>
#include <Modules/Math/share.h>

namespace SCIRun {
```

(continues on next page)

(continued from previous page)

```
namespace Modules {
namespace Math {

  class SCISHARE SortMatrix : public SCIRun::Dataflow::Networks::Module,
    public Has1InputPort<MatrixPortTag>,
    public Has1OutputPort<MatrixPortTag>
  {
  public:
    SortMatrix();
    virtual void execute();
    virtual void setStateDefaults();


    INPUT_PORT(0, InputMatrix, Matrix);
    OUTPUT_PORT(0, OutputMatrix, Matrix);

    MODULE_TRAITS_AND_INFO(SCIRun::Modules::ModuleFlags::ModuleHasUIAndAlgorithm)
  };
}}}
#endif
```

The important differences in this example are that the namespace and type of ports are different, and to leave the *setStateDefaults()* function without brackets so it can be set in the cc file.

The *SortMatrix.cc* file is a bit different from the previous examples. First, since most of the functionality of the module is in the algorithm files, this file can be very short, yet still have a powerful module. This file, along with the header, is mostly the code that pulls the code from the UI and algorithm together and interacts with SCIRun.

```
#include <Modules/Math/SortMatrix.h>
#include <Core/Datatypes/Matrix.h>
#include <Dataflow/Network/Module.h>
#include <Core/Algorithms/Math/SortMatrixAlgo.h>

using namespace SCIRun::Modules::Math;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Core::Algorithms;
using namespace SCIRun::Core::Datatypes;

/// @class SortMatrix
/// @brief This module sorts the matrix entries
/// into ascending or descending order.

MODULE_INFO_DEF(SortMatrix,Math,SCIRun);

SortMatrix::SortMatrix() : Module(staticInfo_)
{
  INITIALIZE_PORT(InputMatrix);
  INITIALIZE_PORT(OutputMatrix);
}

void SortMatrix::setStateDefaults()
{
  setStateIntFromAlgo(Variables::Method);
```

(continues on next page)

(continued from previous page)

```
}

void
SortMatrix::execute()
{
  auto input = getRequiredInput(InputMatrix);
  if (needToExecute())
  {
    setAlgoIntFromState(Variables::Method);
    auto output = algo().run(withInputData((InputMatrix, input)));
    sendOutputFromAlgorithm(OutputMatrix, output);
  }
}
```

Notice that the algorithm file header; SortMatrixAlgo.h, is included.

Also, most of the code in this file links the algorithm code directly to either the UI (with `setStateIntFromAlgo`, and `setAlgoIntFromState`) or SCIRun (`sendOutputFromAlgorithm`).

This allows most of the code to reside in the algorithm code and makes SCIRun more modular.

The final module header and .cc files are included in the source code in `src/Modules/Examples/`. *SortMatrix.cc* and *SortMatrix.h*.

7.5.4 Module UI Code

We create a simple module UI for the SortMatrix module. The UI consists of a toggle switch to choose between ascending and descending sorting. As before, we create three files: *SortMatrixDialog.ui*, *SortMatrixDialog.h*, and *SortMatrixDialog.cc*, all are in `src/Interfac/Modules/Math/`.

The process and code for this example is similar to the previous example.

The *SortMatrixDialog.h* is virtually identical to the header in the previous example (*Simple Module With UI*), except for the names, as shown here:

```
#ifndef INTERFACE_MODULES_MATH_SortMatrixDIALOG_H
#define INTERFACE_MODULES_MATH_SortMatrixDIALOG_H

#include "Interface/Modules/Math/ui_SortMatrixDialog.h"
#include <Interface/Modules/Base/ModuleDialogGeneric.h>
#include <Interface/Modules/Math/share.h>

namespace SCIRun {
    namespace Gui {
        class SCISHARE SortMatrixDialog : public ModuleDialogGeneric,
            public Ui::SortMatrixDialog
        {
            Q_OBJECT

        public:
            SortMatrixDialog(const std::string& name,
                        SCIRun::Dataflow::Networks::ModuleStateHandle state,
                        QWidget* parent = nullptr);
        };
```

(continues on next page)

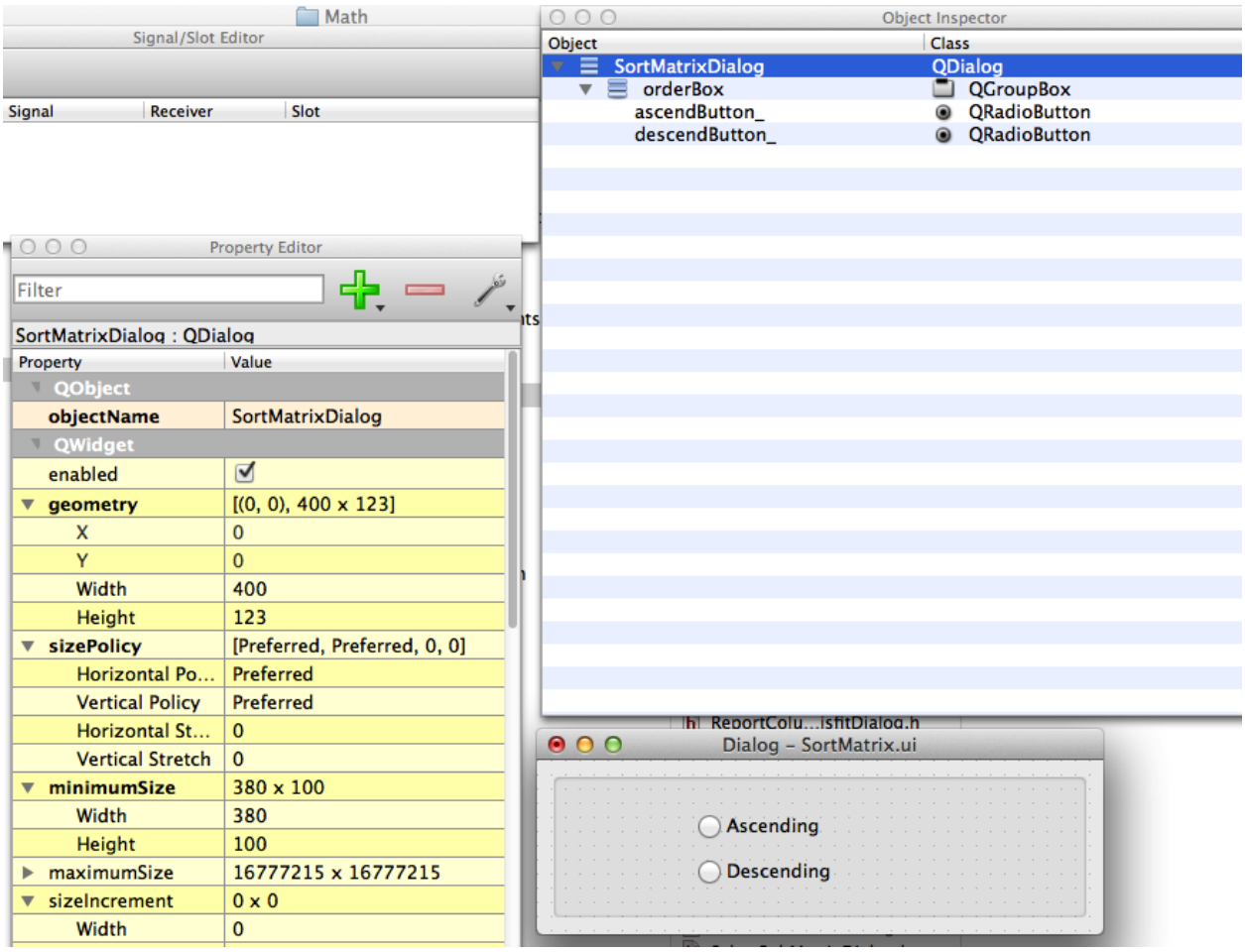


Fig. 7.3: Module interface design file for the SortMatrix module as seen in the Qt editor.

(continued from previous page)

```
}}
#endif
```

SortMatrixDialog.cc is also similar to the dialog .cc file in the previous example (*Simple Module With UI*):

```
#include <Interface/Modules/Math/SortMatrixDialog.h>
#include <Core/Algorithms/Base/AlgorithmVariableNames.h>

using namespace SCIRun::Gui;
using namespace SCIRun::Dataflow::Networks;
using namespace SCIRun::Core::Algorithms;

SortMatrixDialog::SortMatrixDialog(const std::string& name,
                ModuleStateHandle state,
        QWidget* parent/* = nullptr*/)
        : ModuleDialogGeneric(state, parent)
{
        setupUi(this);
    setWindowTitle(QString::fromStdString(name));
    fixSize();

  addRadioButtonGroupManager({ ascendButton_, descendButton_ },
      Variables::Method);
}
```

The key difference in this file is the line that pulls the inputs from the UI using `addRadioButtonGroupManager`. This function pulls a set of radio buttons and creates a toggle switch to assign it to a variable. The order of the button names is important, as that determines the integer values of the variable when it is pressed, i.e., in the order shown, ascending is 0 and descending is 1.

The final module UI code files are included in the source code in `src/Interface/Modules/Examples/`. *SortMatrixDialog.ui*, *SortMatrixDialog.cc* and *SortMatrixDialog.h*.

7.5.5 Module Algorithm Code

The final step in creating the SortMatrix module is to create the module algorithm code. Two files are needed for the algorithm, *SortMatrixAlgo.h* and *SortMatrixAlgo.cc*, which are located in `src/Core/Algorithms/Math/`. The header file contains all the declarations for the functions needed in the algorithm:

```
#ifndef CORE_ALGORITHMS_MATH_SortMatrixALGO_H
#define CORE_ALGORITHMS_MATH_SortMatrixALGO_H

#include <Core/Datatypes/Matrix.h>
#include <Core/Datatypes/DenseMatrix.h>
#include <Core/Datatypes/DenseColumnMatrix.h>


#include <string>
#include <sstream>
#include <vector>
#include <algorithm>
```

(continues on next page)

(continued from previous page)

```
#include <Core/Algorithms/Base/AlgorithmVariableNames.h>
#include <Core/Algorithms/Base/AlgorithmBase.h>
#include <Core/Algorithms/Math/share.h>

namespace SCIRun {
namespace Core {
namespace Algorithms {
namespace Math {

class SCISHARE SortMatrixAlgo : public AlgorithmBase
{

  public:
    SortMatrixAlgo();
    AlgorithmOutput run(const AlgorithmInput& input) const;

    bool Sort(Datatypes::DenseMatrixHandle input,
        Datatypes::DenseMatrixHandle& output, int method) const;

    bool Quicksort(double* input, index_type lo, index_type hi) const;
    index_type Partition(double* input, index_type lo, index_type hi) const;
};
      }}}}
#endif
```

In this algorithm, we have three functions to help implement the sorting algorithm, which are called in the `run` function in *SortMatrixAlgo.cc*.

The *SortMatrixAlgo.cc* file contains the computation code for the module. There are five functions in *SortMatrixAlgo.cc*. The first; `SortMatrixAlgo()` sets up the defaults for the parameters that are set in the module UI. `run()` is the main function of the algorithm and the module. It mostly takes the inputs from the module and sends the data to use in the helper functions. `Sort()` is the main helper function, which further processes the data into a format that be quickly sorted. `Quicksort()` uses the output of `Partition()` to spilt the matrix into smaller and smaller chunks and recursively calls itself until the matrix is sorted.`Partition()` properly places the last entry in the matrix subset, with the values larger after and those lower before, and splits the matrix at that new place for the next iteration. The code is:

```
#include <Core/Algorithms/Math/SortMatrixAlgo.h>
#include <Core/Datatypes/MatrixTypeConversions.h>
#include <Core/Math/MiscMath.h>

using namespace SCIRun;
using namespace SCIRun::Core::Datatypes;
using namespace SCIRun::Core::Algorithms;
using namespace SCIRun::Core::Algorithms::Math;

SortMatrixAlgo::SortMatrixAlgo()
{
  //set parameter defaults for UI
  addParameter(Variables::Method, 0);
}


AlgorithmOutput SortMatrixAlgo::run(const AlgorithmInput& input) const
```

(continues on next page)

(continued from previous page)

```
{
  auto input_matrix = input.get<Matrix>(Variables::InputMatrix);
  AlgorithmOutput output;

  //sparse support not fully implemented yet.
  if (!matrixIs::dense(input_matrix))
  {
    //TODO implement something with sparse
    error("SortMatrix: Currently only works with dense matrices");
    output[Variables::OutputMatrix] = nullptr;
    return output;
  }
  auto mat  = castMatrix::toDense (input_matrix);
  DenseMatrixHandle return_matrix;

  //pull parameter from UI
  auto method = get(Variables::Method).toInt();

  Sort(mat,return_matrix,method);
  output[Variables::OutputMatrix] = return_matrix;
  return output;
}


bool
SortMatrixAlgo::Sort(DenseMatrixHandle input, DenseMatrixHandle& output,
            int method) const
{
  if (!input)
  {
    error("SortAscending: no input matrix found");
    return false;
  }
  //get size of original matrix
  size_type nrows = input->nrows();
  size_type ncols = input->ncols();
  //copy original matrix for processing
  output.reset(new DenseMatrix(*input));
  //pointer to matrix data
  double *data = output->data();

  if (!output)
  {
    error("ApplyRowOperation: could not create output matrix");
    return false;
  }

  size_type n = nrows*ncols;
  //call the sorting functions
  Quicksort(data,0,n-1);

  if (method==1)
```

(continues on next page)

(continued from previous page)

```
  {
    //if set to descending, reverse the order.
    output.reset(new DenseMatrix(output -> reverse()));
  }
  return true;
}

bool
SortMatrixAlgo::Quicksort(double* input, index_type lo, index_type hi) const
{
  //splits matrix based on Partition function
  index_type ind;
  if (lo<hi)
  {
    ind=Partition(input,lo,hi);
    Quicksort(input,lo,ind-1);
    Quicksort(input,ind+1,hi);
  }
  return true;
}

index_type
SortMatrixAlgo::Partition(double* input, index_type lo, index_type hi) const
{
  // places the last entry in its proper place in relation to the other
  // entries, ie, smaller values before and larger values after.
  index_type ind=lo;

  double pivot = input[hi];
  double tmp;
  for (index_type k=lo;k<hi;k++)
  {
    if (input[k]<=pivot)
    {
      tmp=input[ind];
      input[ind]=input[k];
      input[k]=tmp;
      ind+=1;
    }
  }
  tmp=input[ind];
  input[ind]=input[hi];
  input[hi]=tmp;
  return ind;
}
```

This algorithm uses the common inputs defined in *AlgorithmVariableNames.h* with the Variable namespace (`Variable:Method`, `Variable:InputMatrix`, and `Variable:OutputMatrix`). This allows for fewer declarations in the header file and is slightly easier to use. Also of note is that this algorithm is only implemented for dense matrices. This is because some support for sparse matrices hasn't been implemented at the time of writing this tutorial.

The final module algorithm header and .cc files are included in the source code in `src/Core/Algorithms/Examples/` (*SortMatrixAlgo.cc* and *SortMatrixAlgo.h*).

7.5.6 Building and Testing

Building

Once all the files have been created, SCIRun can be built with the new module. Be sure to add all seven files to the appropriate *CMakeList.txt* files in the `src/Modules/Math/`, `src/Interface/Modules/Math/`, and `src/Core/Algorithms/Math/`.

Just add each of the filenames to the appropriate lists within the *CMakeList.txt* file, as shown in the previous examples (*Building and Testing*, *Duplicate the Previous Module*, *Creating a Custom UI*).

When creating new modules, it can be easier to add the code in a piecemeal fashion.

This includes getting SCIRun to build with just the bare minimum of the algorithm code (only `run` with no calls to other functions) then to add the other functions in a step by step manner. This allows for easier debugging and a more systematic process to get the module working. Check *Common Build Errors* for common build problems.

Testing

To make sure that the new SortMatrix module works, create a network with CreateMatrix, SortMatrix, PrintMatrixIntoString, and PrintDatatype as shown in Fig. 7.4. Create any matrix in CreateMatrix.

In PrintMatrixIntoString, change the input to have the number of columns in your input matrix.

In the 4x2 matrix that shown in Fig. 7.4,the format string was: `%4.2g %4.2g %4.2g %4.2g \n`.

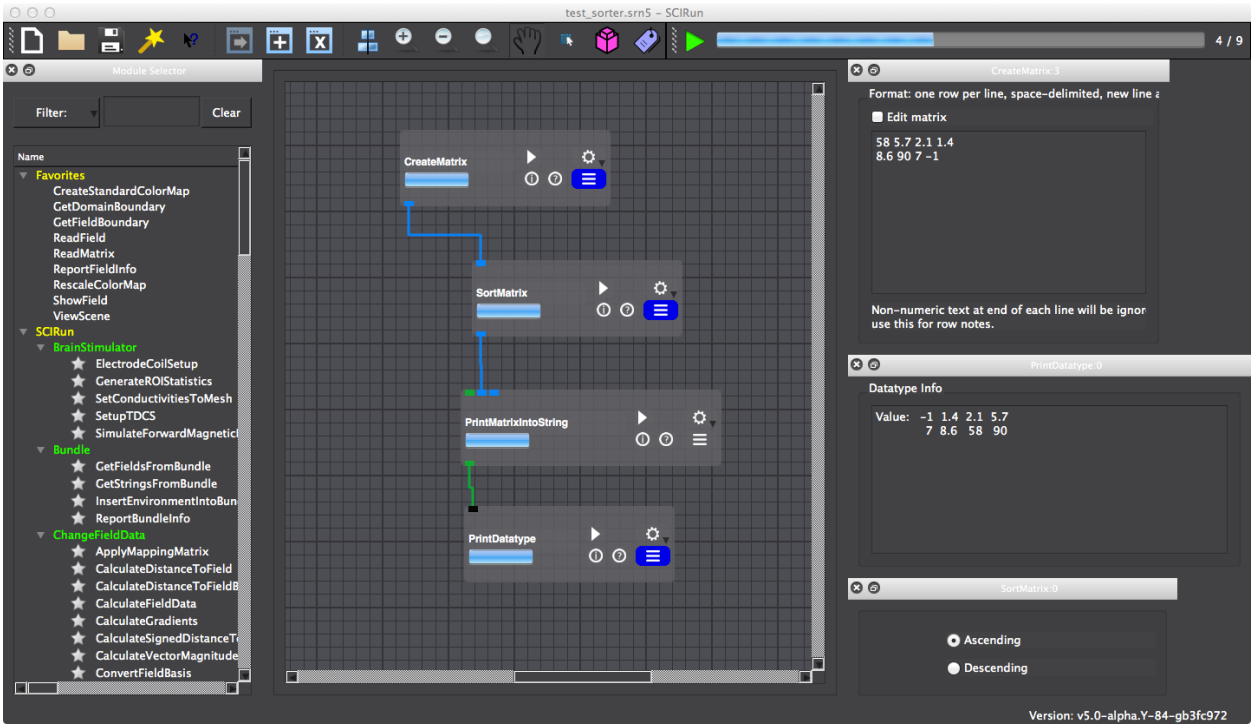


Fig. 7.4: Network for running and testing the SortMatrix module.

Alternatively, the matrix entries can be printed as a list with `%4.2g` (make sure there is a space at the beginning or end of the string).

This network can be used to see the input and output of the SortMatrix module.

If this or another module is not behaving as expected, change the output of some functions and set the output of the module to be some of the intermediate steps, or use `std::cout<< "message" <<std::endl;` to print values as the code runs. Unit Test can also find some bugs in the module code.

7.6 Converting Modules from SCIRun 4

This chapter will walk through the steps necessary to convert a module from SCIRun 4 to SCIRun 5. Converting a module is very similar to creating a new module, as expected. However, there are additional considerations that will be described in this chapter, including a list of common build errors and a list of common changes in code needed for the conversion.

7.6.1 Strategy

The strategy for converting a module from SCIRun 4 to SCIRun 5 is very similar to the strategy of making a new module, which is to start with the basics, and add all the necessary parts piece by piece. This is demonstrated in our previous examples, especially in *Simple Module Without UI* & *Simple Module With UI*.

Another document outlining the steps to convert a module which may be helpful is in the source code: `src/Documentation/Manuals/ModuleConversionSteps.md`. It can also be found on GitHub.

Set up Git Branch

In order for all the hard work of converting a module to be useful for all the users of SCIRun, be sure to use Git and GitHub for version control. Make sure that you have your own fork, sync to the SCIRun repository (*Creating Your SCIRun Fork*). Create a new branch in your fork for each module conversion (*Creating Your SCIRun Fork*) and be sure to commit your changes frequently.

Create a Module Configuration file

Create a module configuration file (*Module Configuration File*) for the module in the `src/Modules/Factory/Config/` directory. It may be easiest to copy an existing file and change it. Be sure that all the names match the appropriate names of other files and fields in other files. It will be easier if the naming convention shown in *Module Configuration File* (name, nameDialog, nameAlgo, etc.) Leave fields blank with `"N/A"`. For now, it can help to only fill in the first section `"module"`, as we will be adding the UI and the algorithm piece by piece. Commit the changes to the local branch for the module.

Create a Module Header file

Most of the modules in SCIRun 4 did not have header files for the module code, so one will need to be created for them. Copy the template header file (*Module Header File*) or another module header file. The header file needs to be in the same location as the module .cc file (next Section). Change all the references to match the name of the module. Change the port names, number, and type. Make sure all functions used in the module code are declared in the header. Commit the changes to the local branch for the module.

Get Module to Build Without Functionality

Getting the module code from SCIRun 4 working in SCIRun 5 can be challenging because there are a number of infrastructure and function changes. For this reason, it can beneficial to start with the basic code and build from there. Most of the SCIRun 4 module code is in the `src/Modules/Legacy/` directory, and can remain there. Make sure the header file is also in this directory.

Once the files are in the correct place, begin by removing all the SCIRun 4 specific code.

If desired, the SCIRun 4 code can be commented out for now instead of deleted, but should be cleaned up before submitting a pull request. Remove any port header file includes, such as: `#include <Dataflow/Network/Ports/FieldPort.h>`, and the `DECLARE_MAKER` function. Remove the class declaration, as the header should already contain the essential declarations. We will add more as needed.

Make sure that the namespaces used are correct (at least two, `Fields/Math/etc` and Networks, are needed as seen in *Module Code File*). Make sure that the module header is included and the other headers included have the correct path. Change the module constructor (`@modulename@::@modulename@()` in the template example) to match the format shown in *Module Code File* with the correct port names. Add the `staticInfo_` variable as in *Module Code File*. Add a blank `setStateDefaults()` function:

```
void @ModuleName@::setStateDefaults()
{

}
```

or, if there is no module UI planned, add empty curly brackets to the header file declaration as discussed in *Module Header File* .

The only code that will carry over is probably the `execute()` and other functions coded in the module, so everything else can be removed. Any helper functions should be moved to an algorithm file later, but for now comment them out. Also comment out the content of the executable for now.

In this state, the three module files (config, header, and .cc file) should be similar to the simplest example (*Simple Module Without UI*), but the execute function will empty for now. Add the header and .cc files the *CMakeList.txt* file in the directory they are in and try to build SCIRun. If there are build errors, check *Common Build Errors* for some ideas for how to correct them. The goal of building the code at this point it to make sure that the code which interacts with SCIRun is working properly before there are other mistakes in the code.

Once SCIRun builds, open it and find the new module. There will probably be a warning about creating a UI because the code is expecting one and there isn't. Make sure all the ports are there and are correctly named (hover the mouse over the port). Commit all changes to the local branch.

If there is no UI planned for this module, add a `false'` input to the module constructor (*Module Code File* & *Module Source Code*). Otherwise, continue with the next step.

Add Module UI

Once the module is building without functionality, a module UI can be added. Begin by copying the UI files from another module with a similar interface, or the template files in `src/Interface/Modules/Template/`. Rename the three UI files (*Module UI Code*) and place them in the appropriate file in `src/Interface/Modules/`. The file names and subsequent function and item names in the UI code should be the same as the module with `Dialog` appended to it.

In the module design file, use the Qt editor to create the UI that is needed by adding and removing widgets as needed (see *Creating a Custom UI* & *Module UI Code*). Make sure that the name of the module and the name of the inputs are correctly named.

Next, modify the dialog header file so that the names of the module and dialog are corrected. There usually isn't anything extra needed with the dialog header. Similarly modify the module dialog cc file.

Now add code to interpret the inputs from the widgets placed in the UI (*Module UI Code*). It may be helpful to look at other modules with similar UIs to determine which functions are needed (see *Creating a Custom UI* & *Module UI Code* for simple examples).

Now that the interface files are created, fill out the Interface section of the module configuration file. Make sure the names are consistent across files. Add all three files to *CMakeList.txt* in the directory that the files are in. Try to build SCIRun. If there are any errors, see *Common Build Errors* for ideas to resolve them. Once SCIRun is built, pull up the module and check the module UI. There will not be any functionality or defaults set, but the look should be correct. If the UI is correct, the UI code should be complete. Commit all changes to the local branch.

Add Module Algorithm Files

Module algorithm code isn't necessary if the module is simple. However using the algorithm class can be an easy way to work with the UI. Therefore, if the module that is being ported does not have algorithm code, consider adding it. If no algorithm code will be added to the module, skip this step.

Copy the algorithm code from SCIRun 4 to the appropriate directory within `src/Core/Algorithms/` (some algorithms have been copied and not ported to SCIRun 5).

If there was no algorithm code in SCIRun 4, copy a similar module algorithm code or the template code found in `src/Core/Algorithms/Template` and modify the names to match the module name with `Algo` added to it. For now, comment out all the code within each of the functions, except any necessary return commands. Some of the functions may need name changes to match the general format in *Module Algorithm Code*.

Now fill out the algorithm section of the module configuration file. Add the algorithm code and header files to the *CMakeList.txt* file in the directory that the algorithm code is in (or possibly the parent directory). Include the algorithm header in the module code, then build SCIRun.

If there are no build errors, start adding the commented out code as described in the next section.

Commit all changes to the local branch.

Add Module and Algorithm Functionality

If all the previous steps are completed, all the files needed for the module have been created and the infrastructure code is working. Next, the functional code of the module will need some work to get working. This step of converting the modules is the most specific to the module and could require the most experience to know the various functions that may have been changed in SCIRun 5. However, there may be some modules that require very little code changes in this step. *Common Function Changes* provides some examples of commonly used functions in SCIRun 4 and the replacement in SCIRun 5. The key to making the module code functional is similar to the other steps, i.e., expose one small piece at a time.

If there is a UI for the module, the state variables defaults need to be set. These need to be set in the `setStateDefaults` function. If there is algorithm code, these default definitions can be passed to the algorithm code, such as the example in *Simple Module With Algorithm*. If there is no algorithm code, the state variables may need to be declared in the module code, as in UI example (*Simple Module with UI*). After setting the state defaults (and building the code), the module UI should display the defaults.

When converting the module code, it may be easier to start on the more standardized code, then work toward the more specific code, as we have been doing through this tutorial. For example, it may be easier to get the input and output calls working (as in *Simple Module Without UI*), and then work on making the output what it needs to. Then, if there is algorithm code for the module, start by getting the algorithm call in the module code working with the algorithm code commented out, then work on the algorithm code. See *Common Function Changes* & *Common Build Errors* for ideas to convert and fix specific functions and pieces of code. Commit all changes to the local branch.

Module Testing

Testing for a new module should occur intermittently while converting the code to make debugging easier, as we described in earlier steps. As you are trying to convert the module, test the module regularly to make sure that the output of module is as expected. Before finishing and submitting the module, test several types of inputs to make sure that the module behaves as intended. Since the module is converted from SCIRun 4, compare the outputs of the different versions.

In addition to making sure that the module works as expected, the module will need to be tested regularly for regression testing. A regression testing network and unit test code is needed for the module. The testing network should show different uses of the module if there are different function. For unit testing see *Creating Unit Tests* for information on creating unit test for the converted module. Commit all test, including the test networks, to the local branch.

Module Documentation

Make sure your module is documented properly in the Git commits and code in addition to the module documentation as described in *Documenting the New Module*.

GitHub Pull Request

With the module fully completed, we can now submit it to be included in the main release of SCIRun using a pull request. Since there was a branch created for the new module, there should be regular commits as the module is ported. For the the pull request, make sure all the changes have been committed to the branch meant for the new module. Now make sure that the branch is up to date with the latest changes in the main branch of SCIRun. To do this, sync your fork and merge the SCIRun master branch as shown in *Creating Your SCIRun Fork*. Make sure the module branch is merge with the master branch and make sure that your local changes are pushed to GitHub. To make a pull request, there is usually a short cut on the main GitHub page of the SCIRun or you can check out the GitHub help page about it. Add some comments to the developers to know what to look for when reviewing the code. If you have changes to make, either that you noticed or requested by the developer, just commit it to the same branch and push it GitHub and the pull request will track the changes until it is merged.

7.6.2 Common Function Changes

SCIRun 4	SCIRun 5	Notes
`if (input.get_rep()==0)`	`if (!input)`	Checks for an empty handle. | Works for all major handle types.
`output = input;|output.detach();`	`FieldHandle output_field(input_field->clone());`	Copy a field and disconnect the data
`output = input;|output.detach();| output->mesh_detach();`	`FieldHandle output_field(input_field->deep_clone());`	Copy a field and disconnect the mesh and data.

7.6.3 Common Build Errors

1.

```
ConvertMeshToPointCloudDialog.h:32:10: fatal error:
      Interface/Modules/Field/ui_ConvertMeshToPointCloudDialog.h' file not found
```

In this instance, the path to the header is misspelled. Check for the correct path and filename. Make sure the file you want to include exists. If the code was converted from SCIRun 4, some of the dependent header files are renamed, combined with other files, or deprecated.

2.

```
/Users/jess/software/SCIRun_mine/bin/SCIRun/Core/AlgorithmsFactoryAlgorithmFactoryImpl_
↪Generated.cc:4:10: error: empty filename
#include <>
         ^
```

```
/Users/jess/software/SCIRun_mine/bin/SCIRun/Core/Algorithms/Factory/AlgorithmFactoryImpl_
↪Generated.cc: 21:43: error: expected namespace name
using namespace SCIRun::Core::Algorithms::;
                                          ^
```

```
/Users/jess/software/SCIRun_mine/bin/SCIRun/Core/Algorithms/Factory/AlgorithmFactoryImpl_
↪Generated.cc: 39:3: error: expected expression
 ADD_MODULE_ALGORITHM_GENERATED(, );
```

This is a result of something wrong with the module configuration file. Check the spelling and syntax of the new file. Specifically, check the quotation characters used, as they may be different (for instance on Mac TextEdit). This may also be caused if the Algorithm and other files have not been added to the `CMakeList.txt` file.

3.

```
/Users/jess/software/SCIRun_mine/src/Interface/Modules/Base/ModuleDialogGeneric.h:71:5:␣
↪warning: 'metaObject' overrides a member function but is not marked
     'override' [-Winconsistent-missing-override]
   Q_OBJECT
   ^
```

```
/usr/local/Cellar/qt/4.8.7_3/lib/QtCore.framework/Headers/qobjectdefs.h:162:32: note:␣
↪expanded from macro 'Q_OBJECT'
virtual const QMetaObject *metaObject() const;
```

Errors that involve Qt and Qt objects deal with the GUI code. Make sure that the GUI name is spelled correctly. Also make sure that the QObject name is set properly.

7.7 Creating Unit Tests

You will at least need a testing network.

7.8 Documenting the New Module

This chapter will describe how to document the modules you add.

7.8.1 Documentation Requirements

You should totally document the modules you add. There is only one required file for a properly documented module. A *ModuleName.md* file is needed along with a symbolic link pointing to this file.

We suggest creating a second branch for the module documentation so that a separate pull request can be made. Instructions to create a new branch can be found in *Creating Your SCIRun Fork*, and be sure to commit your changes frequently.

7.8.2 Creating the Markdown File

Create a file in `docs/_includes/modules/` with the name *ModuleName.md*

A template for the contents is provided here with annotated comments:

```
---
title: MODULE_NAME
category: moduledocs
module:
  category: MODULE_CATEGORY
  package: SCIRun
tags: module
---
<!--- Note: the title must exactly match the module name in the file name -->

# {{ page.title }}

## Category

**{{ page.module.category }}**

## Description

### Summary

<!--- module summary goes here -->

**Detailed Description**

<!-- a more in-depth description goes here -->

{% capture url %}{% include url.md %}{% endcapture %}
{{ url }}
```

7.8.3 Creating the Symbolic Link

Create a symbolic link in `docs/modules` with the same name that points to this new file you created.

OS X/Linux

Execute `ln` command

```
ln -s path/to/original path/to/link
```

For example, if you are in the SCIRun directory, this is what the command might look like:

```
ln -s docs/_includes/modules/ModuleName.md /docs/modules/ModuleName.md
```

Windows

Run Command Prompt as administrator (Right-click on application and click *Run as Administrator*). Execute `mklink` command

```
mklink /path/to/original /path/to/link
```

For example, if you are in the SCIRun directory, this is what the command might look like:

```
mklink docs/_includes/modules/ModuleName.md /docs/modules/ModuleName.md
```

7.8.4 Build Website Locally

To make sure everything is working properly and the module documentation looks right, you can build the website locally.

You will need to have Jekyll installed. Once you have confirmed Jekyll is installed, execute the following command in the `/docs` directory.

```
bundle exec jekyll serve
```

This will build the site as well as make it available on a local server on the server address provided on the command line. If everything was done correctly, the documentation for your new module should be available under Modules and the category provided in *ModuleName.md*. (`/SCIRun/modules.html/#ModuleName`)

Once you have confirmed everything is correct. Be sure to push your changes to GitHub and make a pull request using the same instructions from *GitHub Pull Requests* .

CHAPTER

EIGHT

BIOPSE

8.1 BuildFESurfRHS

Calculates the divergence of a vector field over a surface. It is designed to calculate the surface integral of the vector field (gradient of the potential in electrical simulations).

Builds the surface portion of the RHS of FE calculations where the RHS of the function is GRAD dot F.

Detailed Description

Input

- A FE mesh with field vectors distributed on the elements (constant basis).

Output

- The Grad dot F

CHAPTER

NINE

BRAIN SIMULATOR

9.1 SimulateForwardMagneticField

Calculation of magnetic field at given detector points due to specified dipoles.

Detailed Description

The modules calculates the magnetic field due to given dipoles at specified detector positions based on the Biot-Savart law.

CHAPTER

TEN

BUNDLE

10.1 GetFieldsFromBundle

This module retrieves a **field** object from a bundle.

Detailed Description

This module retrieves a **field** object from a bundle by specifying the name under which the object is stored in the bundle. The module has three output ports that each can be programmed to retrieve a specific field object.

There are two ways of specifying the name of the **field** object. Firstly one can enter the name of the object in the entry box on top of the menu, secondly one can execute the module, in which case a list of all objects of the **field** is generated. By selecting the one that one wants on the output port the object can be retrieved from the bundle.

The first bundle output port generates a copy of the input bundle and can be used to attach a second module that retrieves data from the bundle.

10.2 GetMatricesFromBundle

This module retrieves a **matrix** object from a bundle.

Detailed Description

This module retrieves a **matrix** object from a bundle by specifying the name under which the obect is stored in the bundle. The module has three output ports that each can be programmed to retrieve a specific **matrix** object.

There are two ways of specifying the name of the **matrix** object. Firstly one can enter the name of the object in the entry box on top of the menu, secondly one can execute the module, in which case a list of all objects of the **matrix** is generated. By selecting the one that one wants on the output port the obect can be retrieved from the bundle.

The first bundle output port generates a copy of the input bundle and can be used to attach a second module that retrieves data from the bundle.

10.3 GetStringsFromBundle

This module retrieves a **string** from a bundle.

Detailed Description

This module retrieves a **string** object from a bundle by specifying the name under which the obect is stored in the bundle. The module has three output ports that each can be programmed to retrieve a specific **string** object.

There are two ways of specifying the name of the **string** object. Firstly one can enter the name of the object in the entry box on top of the menu, secondly one can execute the module, in which case a list of all objects of the **string** is generated. By selecting the one that one wants on the output port the obect can be retrieved from the bundle.

The first bundle output port generates a copy of the input bundle and can be used to attach a second module that retrieves data from the bundle.

10.4 InsertEnvironmentIntoBundle

Collects the current environment variables into a bundle.

Detailed Description

InsertEnvironmentIntoBundles reads all the environment variables used in the current SCIRun session and saves them into a bundle for use in the SCIRun Network. It saves each environment variable as a string with the same name as the variable and the contents of the value of the variable. This module is useful in scripting SCIRun because parameters can be set outside SCIRun and passed into SCIRun, e.g. paths to directories were data is stored. Use GetStringFromBundle to read the environment variables saved into the bundle.

10.5 InsertFieldsIntoBundle

This module inserts a **field** object into a bundle.

Detailed Description

This module inserts a **field** object into a bundle. In the GUI of this module a name can be specified for the object. The latter will be used to catalogue the object inside the bundle. This module allows for three **field** objects to be inserted at the same time. In the GUI a name can be specified for each of the **field** objects by clicking on the tab that corresponds to the input port where the dataflow object is located.

10.6 InsertMatricesIntoBundle

This module inserts a **matrix** object into a bundle.

Detailed Description

This module inserts a **matrix** object into a bundle. In the GUI of this module a name can be specified for the object. The latter will be used to catalogue the object inside the bundle. This module allows for three **matrix** objects to be inserted at the same time. In the GUI a name can be specified for each of the **matrix** objects by clicking on the tab that corresponds to the input port where the dataflow object is located.

10.7 InsertStringsIntoBundle

This module inserts a **string** object into a bundle.

Detailed Description

This module inserts a **string** object into a bundle. In the GUI of this module a name can be specified for the object. The latter will be used to catalogue the object inside the bundle. This module allows for three **string** objects to be inserted at the same time. In the GUI a name can be specified for each of the **string** objects by clicking on the tab that corresponds to the input port where the dataflow object is located.

10.8 ReportBundleInfo

This module lists all the objects stored in a bundle.

Detailed Description

This module lists all the objects stored in a bundle. The names of all the objects are listed as well their types.

CHAPTER

ELEVEN

CHANGE FIELD DATA

11.1 ApplyMappingMatrix

Apply a mapping matrix to project the data from one field onto the mesh of another field.

Detailed Description

Mapping data from a **source field** to a **destination field** can be done in a two stage process. First one builds a mapping matrix that describes which linear combination of data values of the source field needs to be taken to form a value in the destination field and secondly one multiplies the data vector of the source data with this matrix to obtain the destination data vector. This module accomplishes the second stage of this process. In order to build a mapping matrix use the module *BuildMappingMatrix* in the MiscField category. The reason for splitting this process is to improve performance of the mapping.

A second advantage of calculating the mapping matrix is that multiple mapping matrices can be built, which then can be multiplied to create the mapping matrix that spans a series of mesh operations.

11.2 CalculateDistanceToField

This module computes the minimum distance between an object (field or mesh) and the nodes in field or the center of the elements.

Detailed Description

This module computes the minimum distance between a node or an element (center of element) and a mesh. In case the object field is a point cloud the module computes the distance to the closest node, in case the object mesh is a curve it will find the closest distance to the curve, etc. The result is a field where the scalar values refer to the distances to the object field. The first input of this module is the field at which the distance field needs to be calculated, the second input port defines the field the distance to which needs to be calculated.

If the original field has the data located on the nodes of the mesh, these nodes are used to calculate the distance to the field object, if the data is located on the elements the distance to the center of the elements is calculated. If no data is stored in the field, the distance to the nodes is calculated.

11.3 CalculateDistanceToFieldBoundary

This module computes the minimum distance between the boundary of the field and the nodes in field or the center of the elements.

Detailed Description

This module computes the minimum distance between the boundary of the field and the nodes in field or the center of the elements. The first input of this module is the field on which we want to calculate the distance, the second input port is the field/mesh the distance to which we want to compute. If the input field has data contained on the nodes, the distance to the nodes of the mesh will be calculated, if the data is located at the elements, the distance to the center of the element will be calculated. If the input field contains no data, the distance to its nodes will be calculated.

11.4 CalculateFieldData

This module calculates a new value for each value in the Field data based on a user defined function. This function is based on a series of variables that is available for each data location. Once the function is defined, the module will walk through each data value and apply the function.

Detailed Description

This module allows the computation of a new scalar, vector or tensor value for each data location in the Field. The user defined function can depend on a number of variables that are defined for each location:

11.4.1 Input Variables

1. `DATA`: This is the current value stored in the Field (either on the element or the node location).
2. `X,Y,Z`: Cartesian coordinates of the node or element (center of the element)
3. `POS`: Vector with Cartesian coordinates of the node or element
4. `A,B,C,...`: Input from additional Matrix ports. The input Matrix can have either 1 row or the same number of rows as there are values in the Field. In case the Matrix has one value this value is the same for each data location, in case it has multiple values the module iterates of the values in the same way it iterates over the data values of the Field. The Matrix input can have either 1 column, 3 columns, 6 or 9 columns. In case the Matrix has 1 column values are assumed to be scalar values, in case the Matrix has 3 columns it is assumed to contain vector values and in case it has either 6 or 9 columns it is assumed to be a tensor value. A 6 valued tensor is defined as xx, xy, xz, yy, yz, and zz.
5. `INDEX`: The index number of the element or node.
6. `SIZE`: The number of elements or nodes in the Field (depends on the input Field mesh type).
7. `ELEMENT`: Special access variable to access properties of the element. Currently only length, area, and volume are available to be called on this entity. In case one is iterating over the nodes, the node point is assumed to be the element, in case one is iterating of the elements, this variable is referring to the full element.

11.4.2 Output Variable

The output needs to be stored in the variable `RESULT`.

11.4.3 Available Functions

A list of available functions is available in the GUI of the module. The *Parser Help* brings up a list of available functions to do scalar/vector/tensor algebra and to view the functions that can be applied to the `ELEMENT` variable.

11.4.4 Input Ports

The first input is the Field whose data needs to be recalculated using a function. The second port is an optional port that allows the user to script the module with a user defined input function. This function will override the function entered in module GUI. The third and next ports are used to import a Matrix. The first port corresponds to Matrix A, the next to Matrix B and so on. These ports can be used to do algebra with values stored as a Matrix or can be used to enter scriptable scalar/vector/tensor values that can be defined elsewhere.

11.4.5 Output Port

The module has one output port that has the newly defined values.

11.4.6 Example Functions

Suppose one wants to set the data values to a certain value:

```
RESULT = 2;
```

This will set every data value inside the Field equal to the value 2.

Similarly one can set the data value to a value specified inside the first Matrix on the input ports:

```
RESULT = A;
```

If the Matrix contains only one value each data point is set to that value, if it contains the same number of values as datalocations, it will map each value in the Matrix to one value in the Field.

One can as well query the positions of the data point:

```
RESULT = X+Y;
```

This will store X+Y in each data location.

This same module can be used as well to generate vector or tensor data:

```
RESULT = Vector(X,Y,cos(A));
```

This will take the X, Y, position and the cos applied to the values in the Matrix A to create a new vector.

One can reuse the value that are there as well:

```
RESULT = DATA+A+B*C;
```

11.4.7 Output Data Type

As the function is parsed using the compiler, the output type cannot be guessed by the module, hence it needs to be set by the user to the correct data type.

11.5 CalculateGradients

This module computes the derivative of a scalar field and output it as a vector field.

Detailed Description

The CalculateGradients module computes the derivative of a scalar field and converts it to a vector field. The gradient is the derivative of three dimensions; the first component in the X direction, the second component in the Y direction, and the third component in the Z direction grid, computing the general direction of "flow" at a specific point on the geometry.

Not all field types are supported by the CalculateGradients module. At this time only derivatives of TetVolMesh and LatVolMesh are supported. The CalculateGradients module has no GUI. If CalculateGradients recieves as input a Field type it does not support, an error message appears in the Error frame of the SCIRun environment.

11.6 CalculateInsideWhichField

This module detects whether a node or a cell is inside any of the object fields. The output field will be indexed according to the object it is in, starting with index 1. Index 0 is reserved for the part of the field not in any of the object fields.

Detailed Description

This module detects whether a node or a cell is inside any of the object fields. The module uses the location of the node or the center of the element to test whether a node or element is inside any of the object fields. The output field will be indexed according to the object it is in, starting with index 1. Index 0 is reserved for the part of the field not in any of the object fields. The module has two options in the GUI, the first one determines whether the detection of the position has to be done for nodes or for elements and the second option allows the user to change the type of field that is generated, the output data will be cased to the selected output type. Also there are options to choose the sampling scheme and the sample points that need to be inside the field.

11.7 CalculateSignedDistanceToField

Creates a field with distances to a closed volume; inside the closed volume the field is positive and negative outside.

Detailed Description

This function needs an object field that is a volume or a closed surface. When inside the object the distance field is positive, when outside it is negative. The module will fail for not close surfaces as one cannot define inside and outside. Similarly this function will not work for objects that are lines or points. Use the *CalculateDistanceToField* module for these objects.

11.8 CalculateVectorMagnitudes

Given a field of vectors, this module will calculate the magnitude of each vector.

Detailed Description

Input: This requires a field with vectors at node or element locations.

Output: This will output a field with a scalar at each node or element location. The scalar will be the vector magnitude.

11.9 ConvertFieldBasis

ConvertFieldBasis can modify the location of data in the input field.

Detailed Description

Upon execution the UI will display the input field name and the current location the data values for the that field are located.

The output field's new data value location is selected from a list of available basis types. The output field will contain the same geometry as the input field with the data value storage at the newly specified location. All output data values are reset to 0.

There is currently support for three basis functions within SCIRun:

1. **None**, meaning that there is no data associated with the field and that no interpolation is supported.
2. **Constant** basis, when data is associated with the elements of a field. This data is not interpolated but is constant within each element and non-continuous at element boundaries.
3. **Linear**, the data is associated with the nodes of a field. Any interpolation will be done linearly within elements (but not across element boundaries).

11.10 ConvertFieldDataType

ConvertFieldDataType is used to change the type of data associated with the field elements.

Detailed Description

ConvertFieldDataType allows the user to change the datatype at each field data_at location. This module also attempts to convert the data in the field to the new type, if possible. If no conversion is possible, the output field will contain default zero values. Note that conversion from a large datatype to a smaller datatype usually results in a loss of precision.

11.11 ConvertIndicesToFieldData

This module takes a field with inidices and puts the data indexed from a matrix in the field.

Detailed Description

This module takes a field with inidices on the nodes or elements of the field and assigns the data indexed from a matrix to the field.

11.12 CreateFieldData

This module calculates a new value for each value in the field data based on a user defined function. This function is based on a series of variables that is available for each data location. Once the function is defined, the module will walk through each data value and apply the function.

Detailed Description

This module allows the computation of a new scalar, vector or tensor value for each data location in the field. This module is closely related to *CalculateFieldData*, however it does not require any data to be present in the field and the data already in the field is ignored. This module also allows the user to state whether he or she wants to calculate data at the elements or at the nodes of the mesh.

The user defined function can depend on a number of variables that are defined for each location:

11.12.1 Input Variables

1. `X,Y,Z`: Cartesian coordinates of the node or element (center of the element)
2. `POS`: Vector with cartesian coordinates of the node or element
3. `A,B,C,...`: Input from additional matrix ports. The input matrix can have either 1 row or the same number of rows as there are values in the field. In case the matrix has one value this value is the same for each data location, in case it has multiple values the module iterates of the values in the same way it iterates over the data values of the field. The matrix input can have either 1 column, 3 columns, 6 or 9 columns. In case the matrix has 1 column values are assumed to be scalar values, in case the matrix has 3 columns it is assumed to contain vector values and in case it has either 6 or 9 columns it is assumed to be a tensor value (A 6 valued tensor is defined as xx, xy, xz, yy, yz, and zz).
4. `INDEX`: The index number of the element or node.
5. `SIZE`: The number of elements or nodes in the field. (Depending on the input field type)
6. `ELEMENT`: Special access variable to access properties of the element. Currently only length, area, and volume are available to be called on this entity. In case one is iterating over the nodes, the node point is assumed to be the element, in case one is iterating of the elements, this variable is referring to the full element.

11.12.2 Output Variable

The output needs to be stored in the variable `RESULT`.

11.12.3 Available Functions

A list of available functions is available in the GUI of the module. Press on the button available functions to obtain a full overview of the current available functions to do Scalar/Vector/Tensor algebra and to view the functions that can be applied to the `ELEMENT` variable.

11.12.4 Input Ports

The first input is the field whose data needs to be recalculated using a function. The second port is an optional port that allows the user to script the module with a user defined input function. This function will override the function given in the GUI of the module. The third and next ports are used to import a matrix. The first port corresponds to matrix A, the next to matrix B and so on. These ports can be used to do algebra with values stored as a matrix or can be used to enter scriptable scalar/vector/tensor values that can be defined elsewhere.

11.12.5 Output Port

The module has one output port that has the newly defined values.

11.12.6 Output Data Type

As the function is parsed using the compiler, the output type cannot be guessed by the module, hence it needs to be set by the user to the correct data type.

11.13 GenerateNodeNormals

Make a new vector field that points to the input point

Detailed Description

This creates a new vector field containing the same geometry and data location as the input field. It works in two modes. If the input PointCloudField contains only one point, all of the vectors in the output field will be attracted to that input point. If the input PointCloudField contains two or more points, the first two points will be used as a line and all the vectors in the output field will be attracted to that line. If the input field contains scalar values, the vectors in the output field will be scaled by those values. Otherwise they will all be normalized.

11.14 GetFieldData

This will get the data (scalar, vector, tensor) associated with the nodes or the elements of a Field and put them in a Matrix or Nrrd with the first entity corresponding to the first matrix entity.

Detailed Description

11.14.1 Input Port

A Field with data on the nodes or elements.

11.14.2 Output Ports

A Matrix, Nrrd, or complex Matrix with the corresponding data.

11.15 MapFieldDataFromElemToNode

This module computes the minimum distance between an object (field or mesh) and the nodes in field or the center of the elements.

Detailed Description

This module computes the minimum distance between a node or an element (center of element) and a mesh. In case the object field is a point cloud the module computes the distance to the closest node, in case the object mesh is a curve it will find the closest distance to the curve, etc. The result is a field where the scalar values refer to the distances to the object field. The first input of this module is the field at which the distance field needs to be calculated, the second input port defines the field the distance to which needs to be calculated.

If the original field has the data located on the nodes of the mesh, these nodes are used to calculate the distance to the field object, if the data is located on the elements the distance to the center of the elements is calculated. If no data is stored in the field, the distance to the nodes is calculated.

11.16 MapFieldDataFromNodeToElem

This module computes the value of the elements based on the data from the adjoining elements. This module supports various operations to map node data to element data.

Detailed Description

Supported methods are:

- AVERAGE - Compute the average of adjoining nodes
- MIN - Compute the minimum value of adjoining nodes
- MAX - Compute the maximum value of adjoining nodes
- SUM - Compute the sum of the adjoining nodes
- INTERPOLATE - Compute a weighted average of the adjoining nodes

11.17 MapFieldDataFromSourceToDestination

This moduled takes a field and finds data values for the destination geometry and outputs the resulting field.

Detailed Description

This modules takes 2 Fields as input:

1. **Source** contains geometry and data values.
2. **Destination** contains geometry only.

This module will find the corresponding data value in the **Source** for each node in the **Destination** field. The data values can be stored at the nodes, cells, edges, or faces of the mesh at either the input or output ports. If the geometries of the input ports are different, the module will calculate data values for the Destination by interpolation.

This module returns 1 output Field:

1. **Remapped Destination** contains the **Destination** geometry with the **Source** input.

11.17.1 GUI Options

The user can select the option of choosing the closest element (**Constant** option) when a data value cannot be interpolated. Otherwise, **Linear** interpolation can be used. The **Maximum Distance** option indicates the maximum distance between points of interpolation and data values. The default for unassigned values can be numeric or NaN.

11.18 MapFieldDataOntoElements

This module maps data from one mesh to another mesh. The output mesh will have the data located on the elements.

Detailed Description

This modules can interpolate a scalar field, or calculate and interpolate the gradient, gradientnorm, or flux.

The first input is the **source** field. The second input allows you to specify the **weights** of the interpolation.

The output is the **target** field containing the interpolated data on the elements.

The GUI options allow you to change if the data is interpolated or taken from the closest element. You can also change how the sampling works within each element.

11.19 MapFieldDataOntoNodes

This module maps data from one mesh or point cloud to another mesh or point cloud. The output mesh will have the data located at the nodes.

Detailed Description

This modules can interpolate a scalar field, or calculate and interpolate the gradient, gradientnorm, or flux.

The first input is the **source** field. The second input allows you to specify the **weights** of the interpolation.

The output is the **target** field containing the interpolated data on the elements.

11.20 MapFieldDataOntoNodesRadialbasis

Maps data centered on the nodes to another set of nodes using a radial basis.

Detailed Description

The first input field is the source field and the second input is the target field. The fields should be formatted for the data to be set on the nodes.

The gui options allow you to set a maximum distance over which to the data will be interpolated. Everything outside that range will be set to the default outside value.

11.21 RegisterWithCorrespondences

This module allows you to morph using a thin plate spline algorithm one point set or mesh to another point set or mesh. It also has the option for a rigid transformation. Both options require correspondence points from both point sets. This module also requires SCIRun to be compiled with LAPACK or Blas.

Detailed Description

- InputField: This will read the node locations from an input mesh that is to be transformed.
- Correspondences1: This reads the node locations from a field of the correspondence points in the new coordinate system
- Correspondences2: This reads the node locations from a field in the same coordinate system as the InputField

11.22 RemoveUnusedNodes

Removes unused nodes from an input mesh.

Detailed Description

Extracts element and node data from the input mesh. Passes through the element data and determines which nodes are being used to define the elements. Any nodes not used to define the elements is eliminated.

Can only be done on unstructured meshes with linear elements.

11.23 SetFieldData

This module allows you to set the scalar, vector, or tensor entries of an array or single column matrix to the nodes or elements of a mesh. The module checks the size of the array and compares it to the number of elements and nodes in order to determine where to put the data. The first entry in the array is applied to the first location on the mesh and this continues sequentially.

Detailed Description

The first input takes a field and it expects a mesh or a point set.

The second input takes an array of data or matrix.

The third input takes NRRD data: this can be used instead in place of the matrix.

11.24 SetFieldDataToConstantValue

This module sets field data to a given scalar value on a new output field based on the input field geometry.

Detailed Description

This module sets field data to a scalar value provided by the user in the GUI on a new output field created with the same geometry as the input field. The field **basis** type can be changed in the output field, or left the same as the input field (default). The scalar value can be set to any type supported by SCIRun (char, short, unsigned short, unsigned int, int, float, double), or left the same as the input field (default).

11.25 SmoothVecFieldMedian

This function smooths vectors assigned to the elements of a mesh using a median filter. The filter creates a neighborhood by asking for elements that share faces two levels deep. The current vector is replaced by the vector with the median vector angle.

Detailed Description

Input requires a mesh with vectors assigned to the elements. The output produces the same mesh with a smooth vector field based on the median filter of the vector angle.

11.26 SwapFieldDataWithMatrixEntries

This module adds and removes data from a field.

Detailed Description

This module performs two simultaneous operations.

First the **Input Field** is split into its mesh part and its data values. The data values are packaged up and passed to the **Output Matrix** port.

Second the input field geometry is joined with the data from the **Input Matrix**, and the result is passed out on the **Output Field** port. If the input matrix is not present, then the second operation is not performed and the Output Field is the same as the Input Field.

The format of the **Output Matrix** will be a column matrix if the input field was of scalar type. It will be an Nx3 matrix if the input field contained vectors. If the field contained tensors it will be Nx9 matrix, where the tensor is flattened out in left to right, top to bottom order. Column 0 contains (0, 0), column 1 contains (0, 1), column 2 contains (0,2), column 3 contains (1, 0), etc.

The **Input Matrix** should have the same number of values as the field where the values are to be stored. The type of the Intput Field is preserved in the Output field as well. So for instance if the input field is a vector field, the Output Field will also be a vector field, and the input matrix should be an Nx3 matrix where N is equal to the number of elements to be filled in.

CHAPTER

TWELVE

CHANGE MESH

12.1 AlignMeshBoundingBoxes

Scales, translates, and deforms an *Input* field to a defined *Alignment* field based on the volumetric bounding boxes that encompass each of these fields.

Detailed Description

The module extracts the center and size dimensions of each field. The center of the input field is reassigned to center of the alignment field. The size dimensions of the input field are likewise matched to those of the alignment field, proportionally reducing the Euclidean distance between nodes of the input field. The result is a transformed input field that shares the same dimensions and position as the alignment field (without rotation).

12.2 CalculateMeshNodes

This module allows the computation of a new position for each node in the input field. The user defined function can depend on a number of variables

Detailed Description

Overview

12.2.1 Input Variables

1. `DATA`: This is the current value stored in the field (either on the element or the node location). The value is only available if the data is located on the nodes.
2. `X,Y,Z`: Cartesian coordinates of the node or element (center of the element)
3. `POS`: Vector with Cartesian coordinates of the node or element
4. `A,B,C,...`: Input from additional matrix ports. The input matrix can have either 1 row or the same number of rows as there are values in the field. In case the matrix has one value this value is the same for each data location, in case it has multiple values the module iterates of the values in the same way it iterates over the data values of the field. The matrix input can have either 1 column, 3 columns, 6 or 9 columns. In case the matrix has 1 column values are assumed to be scalar values, in case the matrix has 3 columns it is assumed to contain vector values and in case it has either 6 or 9 columns it is assumed to be a tensor value (A 6 valued tensor is defined as xx, xy, xz, yy, yz, and zz).
5. `INDEX`: The index number of the element or node.
6. `SIZE`: The number of elements or nodes in the Field (depends on the input Field mesh type).

12.2.2 Output Variable

1. `NEWPOS`: The output needs to be stored in the variable NEWPOS.

12.2.3 Available Functions

A list of available functions is available in the GUI of the module. The *Parser Help* button brings up a list of available functions to do scalar/vector/tensor algebra.

12.2.4 Input Ports

The first input is the Field whose node positions need to be recalculated using a function. The second port is an optional port that allows the user to script the module with a user defined input function. This function will override the function given in the GUI of the module. The third and next ports are used to import a matrix. The first port corresponds to matrix A, the next to matrix B and so on. These ports can be used to do algebra with values stored as a matrix or can be used to enter scriptable scalar/vector/tensor values that can be defined elsewhere.

12.2.5 Output Port

The module has one output port that has the newly defined values.

12.2.6 Output Data Type

As the function is parsed using the compiler, the output type cannot be guessed by the module, hence it needs to be set by the user to the correct data type.

12.3 ConvertHexVolToTetVol

Convert a HexVolField into a TetVolField.

Detailed Description

Given a Hex field (or anything that supports the same interface) as input, produce a TetVol as output – each Hex element gets split into 5 Tets. In order to produce consistent splits across faces, we alternate between two different templates for how to do the split. Currently HexVolMesh, StructHexVolMesh, and LatVolMesh fields can be converted into TetVolMeshes.

12.4 ConvertMeshToPointCloud

Convert input mesh to a PointCloudMesh.

Detailed Description

This converts a mesh to a PointCloudMesh (will not work on non-linear basis). If the input field is a PointCloudMesh, the field will be passed through to the output port. Data values and locations are preserved in the transformation.

12.5 ConvertMeshToUnstructuredMesh

Convert a structured mesh into an unstructured mesh for editing. An unstructured mesh is also implicitly irregular.

Detailed Description

This converts a LatVolField into a HexVolField, an ImageField into a QuadSurfField, or a ScanlineField into a CurveField. The structured meshes are not editable, as that operation does not preserve structure. Data values and locations are preserved in the transformation.

12.6 ConvertQuadSurfToTriSurf

Convert a QuadSurfField into a TriSurfField.

Detailed Description

Given a QuadSurfField (or anything that supports the same interface) as input, produce a TriSurfField as output – each Quad element gets split into 2 Tris. In order to produce consistent splits across faces, we alternate between two different templates for how to do the split.

12.7 EditMeshBoundingBox

EditMeshBoundingBox is used to transform the field geometry.

Detailed Description

EditMeshBoundingBox can trasform the field geometry via the UI or a widget attached to a ViewScene window.

The output field is the transformed input field.

The output matrix is the resultant transformation matrix used to transform the field. It can be used to transform other fields in SCIRun.

The UI can change the center and the size of the output field. Check the box next to the output attribute you wish to modify. The 'Copy Input to Output' button will reset the output values to the input values.

To interactively transform the input field you must attach the Widget Port (the middle output port) to the viewer window.

To move the widget: Shift+LeftClick on the edges of the widget frame and move the transformation box around.

To scale the widget: Shift+LeftClick on the spheres located on each face of the widget and drag. This will scale the output field in a direction normal to the face the sphere is located on.

12.8 FlipSurfaceNormals

This modules changes the normal of the face of an element.

Detailed Description

Changes the normal of the face of an element on a surface mesh by reordering how the nodes are ordered in the face definition. It takes a surface of triangles or quads in the field format as an input and outputs the same surface with the normals going in the other direction.

12.9 GetFieldNodes

This module loads in a field and returns its nodes in the form of a matrix.

Detailed Description

Input: a SCIRun field. Output: All the nodes of the input field, represented by an N*3 matrix. Each row gives the x-y-z coordinates of one node.

12.10 ProjectPointsOntoMesh

Project a point cloud onto a mesh

Detailed Description

ProjectPointsOntoMesh transforms individual points of a point cloud onto the surface of a mesh. This is a simple projection that finds the locations on the mesh (surface or point depending on the settings) that are closest to each point in the point cloud and uses those locations as the new coordinates for the respective points in the point cloud.

There are two options for projection; projecting points onto the elements or nodes of the mesh. Projecting onto the elements of the mesh will allow the points to be anywhere on the mesh. Projecting onto the nodes will ensure the new points will a subset of the mesh points.

There are two inputs to the module. The first input (Field) is the point cloud. The second input (Object) is the mesh onto which the points will be projected.

The output of the module is the projected point cloud.

12.11 RefineMesh

Refines an area in a mesh to have elements of smaller size.

Detailed Description

RefineMesh is meant to make a portion or all of the elements of a mesh a smaller size. The refinement effectively divides each element so that the edge lengths are half, yielding four in the case of hex elements and eight in the case of tet elements. This module works on both hex and tet elements but works best on structured meshes.

The inputs of RefineMesh are the field containing the mesh that you wish to refine and a matrix input for the isovalue used to define the refinement region.

The outputs are a field containing the refined mesh and a mapping matrix that maps data from the unrefined mesh to the refined one.

There are several options in the RefineMesh UI that control the refinement strategy and the region of refinement. The first parameter sets the refinement strategy to the default or an expanded region to improve quality. The second parameter defines the portion of the mesh to refine. There must be non-uniform data on the mesh to specify a region to refine. The options are to refine all (no contraint), refine less than isovalue, refine unequal to isovalue, refine greater than isovalue, or refine nothing. The isovalue can be set in the provided field. The most common way to set a refinement region is to create a distance map determine the desired distance to use. *CalculateDistanceToField* can create a distance map to another mesh.

The RefineMesh algorithm expects the input field to contain data values. Data can be created on an input field using the *CreateFieldData* module.

12.12 ReorderNormalCoherently

This module aligns the normals in one direction.

Detailed Description

This module aligns the normals of the meshes in one direction. It works for open surfaces and not for crossing surfaces. If there are any cross surfaces then they need to be isolated and then Reordering needs to be done.

12.13 ResampleRegularMesh

Resample a regular mesh, such as a LatVol, in order to increase or decrease the number of elements and interpolate the data onto the new regular mesh.

Detailed Description

The input is a field containing a regular mesh. The GUI allows you to change the resolution of the output and to change the interpolation method.

12.14 ScaleFieldMeshAndData

Applies a scale factor to the node locations of the data and/or the data itself. The scale factor can be multiplied to nodes by shifting the centroid to zero and then shifting it back. Or the scale factor can be applied without bringing the centroid to zero. The latter will scale the mesh around zero, and the former will scale the mesh about it's centroid.

Detailed Description

Input requires a mesh in a field format with or without data assigned to the nodes or elements. The second and third inputs are optional matrix fields were you can input the scale factor for the mesh and the scale factor for the data.

12.15 SetFieldNodes

Changes the location of the nodes in a mesh.

Detailed Description

SetFieldNodes replaces the nodes in a mesh with nodes provided in a matrix. This handles most mesh types supported by SCIRun, but the simplest application is with tri or quad surface meshes. This module very simply replaces the old nodes with the new ones, without regard for quality of elements or other considerations, so care must be taken to ensure the nodes are in the same order.

The inputs of SetFieldNodes are the original mesh with nodes and connections, and the matrix of new nodes. The matrix must be Nx3, where N is the number of nodes in the original mesh, so that there is an x,y, and z coordinate for each node. The output is the mesh with the new node locations.

12.16 TransformMeshWithTransform

Non-interactive geometric transform of a field.

Detailed Description

This module is used to non-interactively transform the geometry of a field. The transform is passed in to the **Transform Matrix** port as a 4x4 matrix. There is no GUI for this module. The transform is generally computed by a different module, such as *AlignMeshBoundingBoxes*. For interactive geometry transforms, use the *CreateGeometricTransform* module.

Note that for fields containing vectors and tensors, the direction of the data will be altered by the transform.

CHAPTER

THIRTEEN

CONVERTERS

13.1 ConvertMatrixToScalar

Takes a 1x1 matrix, or a 1x1 subset of a matrix, and converts it to a scalar data type.

Detailed Description

13.2 ConvertMatrixToString

This module prints the contents of a matrix into a string.

Detailed Description

For a dense matrix it outputs all the rows and columns of the matrix. For a sparse matrix only the non zero elements are printed.

13.3 ConvertNrrdToField

This module takes a Nrrd and puts the data into a field.

Detailed Description

Nrrd is both a library and a file format. The full documentation can be found at http://teem.sourceforge.net/nrrd/index.html

The Data Location parameter lets you choose where to set the data in the field. The Field Type parameter determines the data type that the field is composed of. The Convert Parity parameter either does not correct, inverts the current parity or converts to a right handed coordinate system(RHS).

Note about RHS The Teem documentation is not specific on how to deal with RHS and LHS. Hence we interpret the information as following: (1) if a patient specific orientation is given, we check the parity of the space directions and the parity of the objective and convert if needed, i.e. either coord parity or space parity is LHS, then we mirror. (2) if ScannerXYZ is given, nothing is assumed about coord parity and space parity, as it is not clear what has been defined. (3) in case SpaceLeft3DHanded is given, we assume space parity is LHS and coord parity is not of importance.

13.4 ConvertNrrdToMatrix

This module takes 3 optional Nrrd input ports and puts the data into a matrix.

Detailed Description

Nrrd is both a library and a file format. The full documentation can be found at http://teem.sourceforge.net/nrrd/index.html

This module may convert the Nrrd data to a ColumnMatrix, DenseMatrix, or SparseMatrix depending on which ports are connected. The 3 input ports are Data, Rows, and Columns, respectively.

13.5 ConvertRealToComplexMatrix

This module takes the real and imaginary components, in separate matrices, and concatenates them such that the resultant matrix contains the real and imaginary component of each component of the matrix.

Detailed Description

A detailed description of this module is not available at this time. For assistance please contact:

scirun-users@sci.utah.edu

13.6 ConvertScalarToMatrix

This module takes a scalar data value and converts it to a 1x1 matrix.

Detailed Description

13.7 SplitFieldIntoNrrdData

This module takes a field and puts splits the data into 3 Nrrd ports.

Detailed Description

Nrrd is both a library and a file format. The full documentation can be found at http://teem.sourceforge.net/nrrd/index.html

This module creates 3 output ports which are Data, Points, and Connections, respectively.

CHAPTER

FOURTEEN

DATA IO

14.1 ReadBundle

This module reads a **bundle** from file.

Detailed Description

This module reads a **bundle** from file. A bundle file has a .bdl extension and can be written with the WriteBundle module. In a .bdl file every component is stored. Hence it can be used to group a lot of different SCIRun objects together and store it in one file.

This module has one input port through which the user can define the name of the file that needs to be read. If this port is connected to a string dataflow object, this name is used instead the one entered in the GUI.

This module has two output ports: the first port contains the **bundle** that was read and the second port contains a copy of the **filename**. Hence, if one wants to annotate the results, this filename can be displayed using the ShowString module.

14.2 ReadField

This module allows the user to load any of the SCIRun supported **Field** file types and then sets that field in the module's output port.

Detailed Description

Upon opening, the ReadField GUI defaults to reading a single file in the directory where the user's SCIRun executable resides, but also allows the user to navigate to the directory that the user sets for their SCIRUN_DATA environment variable through the Directory widget. The default file type is "*.fld". Other supported file types are listed in the Files of type widget; these are generally NRRD, Matlab and mesh types. Files generally contain point cloud, surface or volume geometry such as point cloud points, triangle surface points and elements, tetrahedral mesh points and elements etc.

A typical way to import data into SCIRun is to read in geometry, and then also files containing data, then apply the data using a module like *SetFieldData*.

If the user attempts to read in a file other than a SCIRun supported file type, or uses an incorrect file format, an error message will be logged on the SCIRun module. Clicking the **Set** button will load the file. Clicking the **Execute** button will send the loaded field downstream if other modules are connected to the ReadField module's output port.

The **Time Series** tab allows the user to load files numbered sequentially with a common basename with an optional delay. The playback controls allow the user to play through the files, or step forward and backwards.

14.3 ReadMatrix

This module reads a persistent matrix from a file and outputs that matrix to another module.

Detailed Description

Upon opening, the GUI defaults to the directory that the user sets for their SCIRUN_DATA environment variable. Otherwise, the GUI will default to the directory where the user's SCIRun executable resides.

The current directory defaults to only show files with an .mat extension which helps the user determine the difference between matrix files and other files. However, the ReadMatrix can read-in a file with any extension so long as the data has the correct format.

If the user attempts to read-in a file that is other than a SCIRun supported matrix, or uses an incorrect file format, an error message appears.

14.4 ReadNrrd

This module allows the user to load Nrrd files and then sets that Nrrd data in the module's output port.

Detailed Description

Nrrd is both a library and a file format. The full documentation can be found at http://teem.sourceforge.net/nrrd/index.html

This module is a wrapper around the Nrrd library's function `loadNrrd()`.

Upon opening, the ReadNrrd GUI defaults to reading a single file in the directory where the user's SCIRun executable resides, but also allows the user to navigate to the directory that the user sets for their SCIRUN_DATA environment variable through the Directory widget. The only supported file type is "*.nrrd".

If the user attempts to read in a file other than a SCIRun supported file type, or uses an incorrect file format, an error message will be logged on the SCIRun module. Clicking the **Set** button will load the file. Clicking the **Execute** button will send the loaded field downstream if other modules are connected to the ReadField module's output port.

14.5 WriteField

This module saves persistent field objects received from upstream modules.

Detailed Description

Upon opening, the GUI defaults to the directory that the user sets for their SCIRUN_DATA environment variable.

Otherwise, the GUI will default to the directory where the user's SCIRun executable resides. The user can enter the name of a file that the matrix saves to. The file should have a .fld extension. The default file name is MyField. The user may also choose between a Binary or ASCII file format.

14.6 WriteMatrix

This module saves a persistent representation of a matrix to disk.

Detailed Description

Upon opening, the GUI defaults to the directory that the user sets for their SCIRUN_DATA environment variable.

Otherwise, the GUI will default to the directory where the user's SCIRun executable resides. The user can enter the name of a file that the matrix saves to. The file should have a .mat extension. The default file name is MyMatrix. The user may also choose between a Binary or ASCII file format.

CHAPTER

FIFTEEN

FINITE ELEMENTS

15.1 ApplyFEMCurrentSource

The ApplyFEMCurrentSource module builds the right-hand side (ColumnMatrix) to reflect monopolar and dipolar current sources for finite-element based bioelectric field problems.

15.1.1 Inputs:

15.1.2 Outputs:

The first output, RHS, is the right-hand-side vector to be used with the stiffness matrix. The size of the vector is the number of nodes of the input mesh.

The second output, Weights, is a vector giving the weight of each component of the current source. When dipole sources are used, Weights contains the x/y/z-component of each dipole moments.

Detailed Description

If an input RHS ColumnMatrix is present, this module adds the contributions for the current sources into that matrix (i.e. multiple ApplyFEMCurrentSource modules can be cascaded together). If no RHS is present, this module will generate one. The RHS ColumnMatrix will be of length n, where n is the number of nodes in the finite element mesh.

We assume the mesh will be stored in a TetVolMesh, a HexVolMesh, or a TriSurfMesh.

There are two types of current sources supported by this module: dipoles, and sources-and-sinks.

With dipoles, the user must pass in a PointCloudField into the Sources input port. The position of each PointCloud node corresponds to the location of the dipole, and the corresponding Vector corresponds to the moment of the dipole. If the user wishes to use dipolar sources, they must pass in a PointCloudField AND they must select "dipole" for the Source Model on the UI window.

In contrast to the dipole model, the sources-and-sinks option has several sub-models. To activate any of these, the user must select the "sources and sinks" option from the UI. The values that are used for sources-and-sinks are a combination of: the source and sink indices entered in the UI; a PointCloudField passed into the Source port; and a MappingMatrix. Here is the logic for how those values are used:

1. if we don't have a Mapping matrix, we use the source/sink indices from the UI as node indices from the volume mesh between which one unit of current is passed;
2. if we have a Mapping matrix, but we don't have a Source field, then the source/sink indices refer to the PointCloud and we use the Mapping matrix to get their corresponding volume node indices, and we then pass one unit of current between them;

3. if we have a Mapping matrix AND a Source field, then ignore the source/sink indices from the UI, and assume that the Mapping matrix maps the PointCloud nodes to Volume mesh nodes and that the data values (doubles) from the PointCloud indicate the strength (current density) for each point source.

15.2 ApplyFEMVoltageSource

The module changes/creates the right-hand side (RHS) and forward matrices for voltage (Dirichlet) boundary conditions.

Detailed Description

The ApplyFEMVoltageSource module creates and fills a new ColumnMatrix, according to mesh and dipole parameters in situations where it does not locate a ColumnMatrix on its input that contains an RHS to modify, or when the module finds a ColumnMatrix of a different size than the number of nodes in the mesh.

15.3 BuildFEMatrix

This module sets up a mesh with conductivity tensors for bioelectric field problems.

Detailed Description

The module provides basic means for construction of the FE matrix with linear elements for bioelectric field problems (discretization of Poisson equation for volume conductor problem).

The input field should be a Field (TriSurf, HexVol, or TetVol) of either Tensors or integers, with the data values defined at the elements (constant over each element). If it is a Field of Tensors, each Tensor indicates the local conductivity tensor at that location in the mesh. If it is a Field of Integers, then each integer is an index into a lookup table, where the table contains a set of conductivity tensors.

When using the index/table approach, a **conductivity_table** property must be stored with the Field, where the conductivity table is of type pair<string,Tensor> (i.e. the individual material types can have descriptive names). The BioPSE::Modeling::ModifyConductivities module can be used to generate (as well as to modify) the conductivity_table property. The BioPSE::Forward::IndicesToTensors and BioPSE::Forward::TensorsToIndices modules are useful for converting between the index/table (Field of integer with a conductivity_table Propert) and the full Tensor (Field of Tensors) representations.

BioPSE::Algorithm::BuildFEMatrix::build_FEMatrix performs the actual computation of the matrix. Other modules or any other code may use the algorithm for other kinds of problems involving discretization of the Poisson equation by linear finite elements.

The discretization of the Poisson equation on the supplied mesh is performed by linear finite elements using Galerkin approach.

15.4 BuildFEVolRHS

This module calculates the divergence of a vector field over the volume. It is designed to calculate the volume integral of the vector field (gradient of the potential in electrical simulations).

Then, it builds the volume portion of the RHS of FE calculations where the RHS of the function is GRAD dot F.

Detailed Description

The input is a FE mesh with field vectors distributed on the elements (constant basis).

The output is the Grad dot F.

CHAPTER

SIXTEEN

FLOW CONTROL

16.1 BooleanCompare

Compare one or two matrices using boolean operators, and send different outputs, or quit SCIRun, depending on the result of the boolean comparison.

Detailed Description

The BooleanCompare module will evaluate one or two input matrices and return different results based ont the evaluation. The module will evaluate either the values, size, or norm of each of the matrices with boolean operations, and can return any one of three inputs, as well as quit SCIRun. The inputs will be compared element by element based on the criteria chosen in the Condition option and perform the actions chosen in the Then option if true and Else option if false. Module also send the result of the comparison as the seond output matrix.

Inputs

- MatrixA - (Required) First matrix for comparison.
- MatrixB - (Optional) Second matrix for comparison. If this port is not used, most of the comparsion options will not be possible.
  PossibleOutput - (Optional) Possible matrix to send as an output to the module.

Outputs

- OutputMatrix - The output matrix of the module. This will be identical to one of the three inputs of the module, or will be null depending on the inputs and options of the module.
- BooleanResult - Matrix of 1x1 containing either 1 or 0 indicating the result of the boolean comparison performed by the module.

Options

Any combination of the values (default), size, and norm (except a size and norm) of the first two input matrices can be compared as long as the resulting size of the objects are the same. The size of one matrix can be compared to the values of the other matrix if it has two entries. Likewise, the norm of one matrix can only be compared to the value of another matrix if it is 1x1.

The condition option sets the method of comparing the input matrices. If the operator is true, then the module will result in the action chosen in the Then_Option menu, and if false it will perform the action selected in the Else_Option menu. Operator options are:

- 'A is non-zero' - (default) true if there are any non-zero entries of A .
- 'A and B are non-zero (and)' - true if any corresponding entries of A and B are both non-zero.
- 'Either A or B is non-zero (or)' - ture if either A or B has non-zero entries
- 'A is greater than or equal to B (>=)' - true if entries of A are greater than or equal to B's.'

- 'A is greater than B (>)' - true if entries of A are greater than B's.
- 'A is equal to B (==)' - true if all the entries of A are the same as B.
- 'A is less than B (<)' - true if entries of A are less than B's.
- 'A is less than or equal to B (<=)' - true if entries of A are less than or equal to B's.

There can be ambiguous scenarios with the greater or equal, greater, less, less or equal options in the module if the input matrices have a size of more than one. In these cases, ie, if there are a set of entries which are both less than and greater than, the number of entries and the magnitude of the difference will determine which input is greater or less. Consider comparing the matrix norms to avoid ambiguity of the results. If there is no second matrix input, the first matrix can only be evaluated by non-zero entries (or size or norm) of the matrix.

The results options establishes the output of the module based on the result of the comparison. If the condition returns true, the action indicated in the Then option will be performed, If false, the Else option will be performed. Possible actions are:

- 'Return First Input' - (Then default) returns the first matrix input.
- 'Return Second Input' - returns the second matrix input.
- 'Return Third Input' - returns the third matrix input.
- 'Return null' - (Else default) returns an emplty matrix handle.
- 'Quit SCIRun' - Quit the current instance of SCIRun.

If the second or third inputs are selected for output and no matrix is input into these ports, the module will return null.

16.2 ChooseInput

Choose one of any number of inputs (of any type) to send to the output.

Detailed Description

The ChooseInput module allows users to send any one of several inputs as an output to the module. This module can be used to quickly switch between data sent to other modules. Input Data can be of any type (Field, Matrix, String, etc), and the output will be the type (and the same data) of the chosen input. Choose which input to send by setting the input port number in the module UI. There are no constraints on type or size of the input data, therefore the user should consider the output destination ensure that all the input data to the ChooseInput module is compatible with the destination port.

CHAPTER

SEVENTEEN

FORWARD

17.1 BuildBEMatrix

The module solves a discretized model of Laplace's equation in a 3D volume conductor model using "surface method", given a particular type of boundary condition.

Detailed Description

The specific problem the authors had in mind was the forward problem of electrocardiography which consists of calculating, for a given time instant, the potential distribution generated at the surface of a specified volume conductor due the presence of the selected equivalent sources on a surface inside the conductor (The module should work for other problems which fit the same physical description, but has only been tested for forward electrocardiography.) In surface methods, the different volume conductor regions are assumed to have a constant and isotropic conductivity, and only the interfaces between the different regions are triangulated and represented in the numerical model. This module uses the "Transfer-Coefficient Approach" or "solid-angle method" developed by Barr et al. [BRS77], as extended to include torso inhomogeneities in [SPM86] and with an improved algorithm for computing the solid angles [dM92].

This module requires the triangulated surfaces of all the objects as inputs and creates the forward solution matrix as output. The geometric relationships of the surfaces are defined as described below, as are the boundary conditions to apply.

The number of input fields is two or greater and can be unlimited but one of them must be defined as the "source surface" where the Dirichlet boundary condition is given and another one defined as the "measurement surface" where the quantity of interest is to be calculated. To do this we use a "SetProperty" module for each of these two designated surfaces with `Property` = `in/out` and `Value` = `in` for the "source surface" and `Value` = `out` for the "measurement surface". Insulating boundary conditions (Neumann boundary conditions) are assumed on the outermost surface.

To define the geometric relationships of the various fields, for each of the input fields use a "SetProperty" module with `Property` = `Inside Conductivity` and `Value` = the numerical value of the internal conductivity of the corresponding homogeneous region.

The output is the forward solution matrix. This matrix can be multiplied to a Dirichlet boundary condition on the "source surface" to result in the boundary values on the "measurement surface". This matrix is needed as an input for the modules "TikhonovSVD", "Tikhonov" and "TSVD".

17.2 CalcTMP

Calculates the transmembrane action potentials given the input parameters generated from ECGSim

Detailed Description

17.2.1 Inputs as matrices - These can be generated by ECGSim

- Amplitude
- Depolarization Times
- Depolarization Slope
- Plateau Slope
- Repolarization Slope
- Resting Potential

17.2.2 Output

- Transmembrane potentials for one cardiac cycle.

17.3 CalculateCurrentDensity

Compute the current density vector field

Detailed Description

CalculateCurrentDensity calculates the current density vector field.

Required inputs are the electric field and mesh with conductivities.

Technical note: The current density vector field J is the product of sigma and -del V. The minus sign is added in CalculateCurrentDensity, so the electric field input should be positive (which is the unmodified output of the *CalculateGradients* module).

17.4 InsertVoltageSource

Insert electrodes into a finite element mesh.

Detailed Description

Insert electrodes into a finite element mesh.

CHAPTER

EIGHTEEN

INVERSE

18.1 BuildSurfaceLaplacianMatrix

Transform attributes

Detailed Description

Transform attributes

18.2 SolveInverseProblemWithTSVD

This module solves a linear inverse problem with a truncated singular value decomposition (TSVD).

Detailed Description

One can compute the inverse of an invertible matrix from its SVD. Specifically, an invertible matrix A=U*S*V' has an inverse inv(A)=V*inv(S)*U' (where "inv()" means matrix inverse). Since the matrix S is a diagonal matrix, its inverse is just the scalar inverse of its diagonal elements. However, when A is not invertible or it is ill-conditioned, a more stable solution to the inverse problem may be obtained by omitting the small or zero-valued singular values from the SVD-based matrix inverse computation, which is the method implemented in this module. This solution is mathematically equivalent to solving the inverse problem only in the subspace spanned by the right singular vectors (rows of V) corresponding to singular values that were not discarded.

This method is sometimes referred to as a pseudo-inverse when only the zero-valued singular values are discarded.

18.3 SolveInverseProblemWithTikhonov

SolveInverseProblemWithTikhonov solves the inverse problem as a (weighted) minimum norm problem by applying a Tikhonov L2-norm regularization. It solves the minimization problem:

$\hat{x} = argmin_x \|Ax - Ly\|_2^2 + \lambda \|Rx\|_2^2$

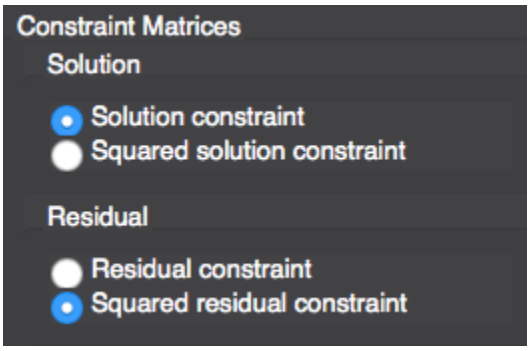
Detailed Description

The module has four inputs, three outputs, and a graphical user interface (UI) where the regularization parameter can be chosen or computed. All inputs can be generated by SCIRun, or created elsewhere or loaded from Matlab files or the Matlab interface.

18.3.1 Input

1. Forward problem matrix, A: This matrix describes the relationship between solution and measured signals $Ax = y$. For example, this matrix can be computed by solving the forward problem e.g. using finite or boundary element methods (see modules in BioPSE Forward, *FiniteElements* category).
2. Solution constraint matrix, R: This matrix is used to constrain the solution. It imposes some a priori knowledge upon the minimization problem to select solutions with specific properties (expressed as a solution covariance e.g. to achieve maximally smooth solutions). If no input is given the identity matrix will be used.
3. Data vector, y: These are the measurement data which have to be provided as matrix which should only contain one column.
   In case of having a time series (multiple columns), it is recommended to use the module *SelectSubMatrix* to iterate over the different time instances (columns). In more detail, for each time instance, *SelectSubMatrix* would provide the data of the current time instance for SolveInverseProblemWithTikhonov to perform an estimation of the inverse solution.
4. Residual constraint matrix, L: This matrix is used to weight the measurements (e.g. by an inverse covariance matrix of the measurement channels). If no input is given, the identity matrix will be used.

IMPORTANT: Both, residual constraint as well as the solution constraint matrix can be used in single matrix form (such as L and R) in the minimization problem or as a squared matrix version (such as $L^T L$ and $R^T R$) respectively. This option must be properly selected by the user in the UI:



18.3.2 Output

1. Inverse solution, $\hat{x}$: computed solution estimate.
2. Regularization parameter, λ: used regularization parameter.
3. Regularized inverse matrix, G: linear inverse operator that gives a solution estimate equation $\hat{x} = Gy$. It actual value depends on the selected formulation (underdetermined or overdetermined) and requires the inversion of a matrix. It is only calculated if this port is connected to another module's input port. The user can select between the formulations using the module GUI (see section *Computation*).

- underdetermined:

$$G = RR^T A^T (ARR^T A^T + \lambda LL^T)^{-1}$$

- overdetermined:

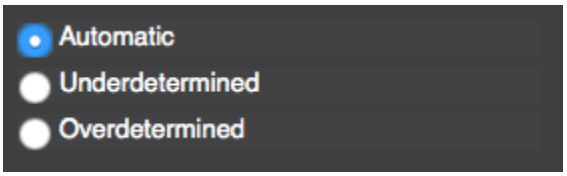
$$G = (A^T L^T LA + \lambda R^T R)^{-1} A^T L^T L$$

18.3.3 Computation

The algorithm has two different ways of computing the solution: the underdetermined and the overdetermined formulations. To choose which one will be solved the module has three options:

- Automatic selection. The algorithm will select underdetermined or overdetermined based on the dimensions of the Forward matrix.
- Manual selection of the undetermined formulation.
- Manual selection of the overdetermined formulation.

These options are presented in the GUI as follows:



Both cases use Gaussian elimination to calculate the unknown $\hat{x}$, but they differ in the equations to solve for:

- underdetermined:

$(ARR^TA^T + \lambda LL^T)b = y, \hat{x} = RR^TA^Tb$

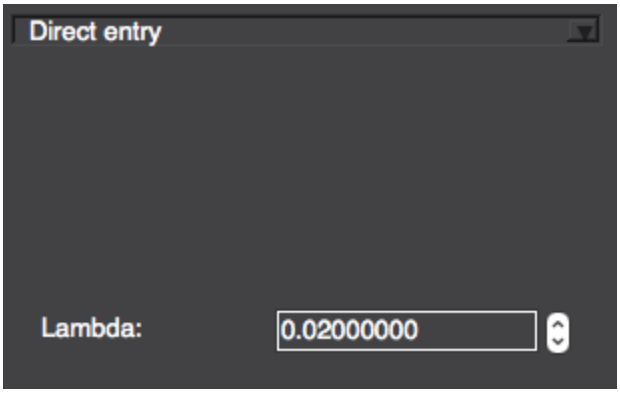
- overdetermined:

$(A^TL^TLA + \lambda R^TR)\hat{x} = A^TL^TLy$

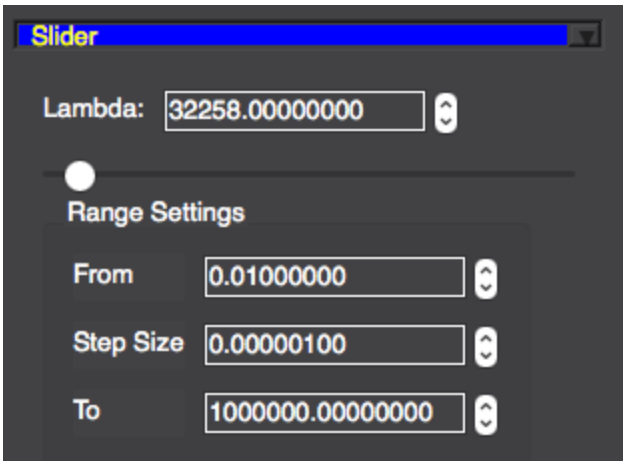
However, If the output port of the inverse operator is connected to another module's input the inverse operator G is used to compute the solution estimate $\hat{x}$ such as $\hat{x} = Gy$.

In both cases the used regularization parameter can be chosen in the UI. The available options are:

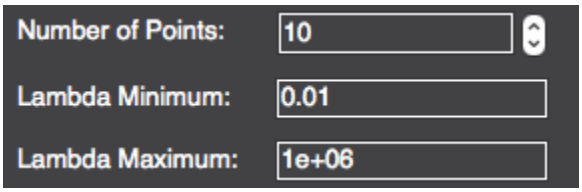
- Fix λ manually.



- Automatic selection of λ: the corner of the L-curve is determined by maximal curvature. A number of regularization parameter points (different λ values) can be specified in the L-curve plot which are than equally distributed over the range ("Lambda Range", see below) of λ's to be used in the L-curve computation.
- Fix λ manually based on the range of regularization parameters (λ) specified by the "Lambda Range" input.



For the options that use the L-curve, the user can define the range and step size in the UI:



18.4 SolveInverseProblemWithTikhonovSVD

This module solves a linear inverse problem with Tikhonov regularization. It uses the singular value decomposition (SVD) or generalized SVD (GSVD) to compute its result.

Detailed Description

This module computes Tikhonov regularization as in SolveInverseProblemWithTikhonov but in a computationally efficient manner for multiple repetitions are needed since it uses the SVD (or GSVD if there is a regularization matrix) decomposition of the forward matrix. Thus, the optimization problem being solved is:

$\hat{x} = argmin_x \|Ax - y\|_2^2 + \lambda \|x\|_2^2$

Where the forward matrix A is expected to be given decomposed in its SVD terms that can be obtained with the module CIBC:Documentation:SCIRun:Reference:SCIRun:ComputeSVD. Note that computeSVD module returns the right singular vectors in rows and not in columns as it is required for SolveInverseProblemWithTikhonovSVD.

The GSVD should be used when the regularization matrix is different than idenity matrix (e.g. approximate Laplacian operator on a meshed volume or surface).

The solution is dependent on a scalar regularization parameter λ that can be specified directly using the User Interface of this module or can be determined automatically using the L-curve method. The automatic determination of the regularization parameter with the L-curve method requires computing several solutions to the inverse problem for a range of regularization parameters.

18.4.1 Input

1. Left Singular Vectors matrix, U.
2. Singular Values vector, S.
3. Right Singular Vectors matrix (vectors in columns), V.
4. Measured ECG vector, y

IMPORTANT NOTE: unlike some other software packages, this module expects the right singular matrix to contain its singular vectors in the columns. That is the matrix V in the decomposition $A = U * S * V^T$.

18.4.2 Output

1. Inverse solution, $\hat{x}$: computed solution estimate.
2. Regularization parameter, λ: used regularization parameter.
3. Regularized inverse matrix, G: linear inverse operator that gives a solution estimate equation $\hat{x} = Gy$. It actual value depends on the selected formulation (underdetermined or overdetermined) and requires the inversion of a matrix. It is only calculated if this port is connected to another module's input port. The user can select between the formulations using the module GUI (see section Computation).

CHAPTER

NINETEEN

MATH

19.1 AddKnownsToLinearSystem

This module deals with solving a linear system `A*u=b` when some values of the vector u is already known. The module will modify the linear system according to the known values.

Detailed Description

This module takes 3 inputs:

1. **LHS Matrix** is an NxN Matrix (must be a SparseRowMatrix).
2. **RHS Vector** is the right hand side vector, an Nx1 Matrix.
3. **X Vector** is an Nx1 Matrix specifying the known variables in the linear system LHS. If the k*th* variable is known, `x(k)` is its value, otherwise `x(k)` should be set NaN.

This module returns 2 outputs:

1. **OutPutLHSMatrix** is an NxN Matrix.
2. **OutPutRHSVector** is an Nx1 Matrix.

The solution:

```
inv(A2)*b2
```

has the same value as x at the non-NaN positions.

19.2 AddLinkedNodesToLinearSystem

AddLinkedNodesToLinearSystem will link nodes in a finite element mesh and solve them as a single value.

Detailed Description

AddLinkedNodesToLinearSystem works with *AddKnownsToLinearSystem* and *SolveLinearSystem* to force nodes in a finite element mesh to solve to the same value. This module is useful in solving for regions with homogeneous field properties, such as with a perfect conductor in an electric field. The algorithm to do this involves combining rows in the stiffness matrix corresponding to the nodes to solve together. The module also creates a mapping Matrix that will map the results of the solved linear system to the original mesh.

19.2.1 Input Ports

This module takes 3 inputs. The first two are the NxN Matrix (LHS) and the right hand vector (RHS), which is a Nx1 Matrix, for a linear system (`Ax=b`) and correspond to the two outputs of the *AddKnownsToLinearSystem* module.

The third input is a matrix (LinkedNodes) representing the data marking the nodes to link on the same mesh used in the *BuildFEMatrix* module. The nodes should be marked with an integer value so that nodes to be linked will have the same value. Any number of linked node sets can be used by using a different value, i.e., independent sets of nodes to be linked must be marked 1,2,3 creating three regions of linked nodes. The remaining nodes should be marked NaN. This can be accomplished be using *MapFieldDataOntoNodes* and using the ***Default Outside Value of NaN*** option and a ***Maximum Distance*** of some value that is small relative to your mesh size. Then use GetFieldData to port the matrix into this module.

19.2.2 Output Ports

This module returns 3 outputs. The first two are the reduced Matrix (OutputLHS) and right hand vector (OutputRHS), which is a Nx1 Matrix, for a linear system to use in *SolveLinearSystem*. The third is a mapping Matrix (Mapping) to restore the data (solution vector from linear system solution) to the proper size. To do this, multiply the mapping matrix and the first output of *SolveLinearSystem*. The resulting vector can then be applied to the original mesh used in the calculation.

19.3 AppendMatrix

This module appends the rows or columns of SecondMatrix, and optionally other matrices, to FirstMatrix.

Detailed Description

This module appends Matrix from the SecondMatrix input port, and optionally other matrices from dynamic Matrix ports, to the base Matrix from the FirstMatrix input port. The user can use the GUI to select whether the rows or the columns of the input Matrix are appended to the base Matrix.

19.4 BuildNoiseColumnMatrix

This module performs the unary matrix operation transpose.

Detailed Description

19.5 CollectMatrices

This module appends/replaces a column or a row in/to a matrix while looping through a network.

Detailed Description

This module appends, replaces, or prepends a column or row of a matrix. This module does one of these operations each time the module is executed. Hence, the appending or replacing operation is in time.

This module has two input ports:

1. base matrix, which is read the first time the module is executed
2. the matrix that needs to be appended is entered through the second input port

When the user wants to restart the network, he or she needs to clean this module as it would continue to collect matrices. To clear the matrices in the buffer of this module, hit the **Clear Output** button in the GUI, which will reset this module. The UI has options to allow the user to select:

Use the GUI to specify the operation which has to be performed each cycle of a network execution: the first column in the GUI specifies whether the module works on rows or columns, the second specifies whether one wants to add or replace the last column or row. The last column specifies whether a column/row is replaced at the start of the matrix (row 0 or column 0) or at the end of the matrix.

19.6 ComputePCA

A Description of this module is not available at this time. For assistance please contact:

scirun-users@sci.utah.edu

Detailed Description

19.7 ComputeSVD

This module computes the singular value decomposition (SVD) of a matrix.

Detailed Description

This module takes a matrix as input, converts it to a dense matrix, computes its SVD, and outputs its three SVD matrices. The three matrices are the left singular vectors (stored in the **columns** of 'LeftSingularMat'), singular values (in decreasing order, stored in the column matrix "SingularVals"), and the right singular vectors (stored in the **rows** of "RightSingularMat"). The number of elements in "SingularVals" is equal to the smallest dimension of the input matrix.

This module requires a version of SCIRun built with LAPACK support.

19.8 ConvertMatrixType

This module casts a matrix.

Detailed Description

Displays type of the input matrix.

Allows you to let the matrix stay the same and just pass through, or cast it to a ColumnMatrix, DenseMatrix or SparseRowMatrix.

19.9 CreateComplexMatrix

This module allows users to input, manually, complex numbers that are then cast into the complex data type. Only one data entry possible per line.

Detailed Description

A detailed description of this module is not available at this time. For assistance please contact:

scirun-users@sci.utah.edu

19.10 CreateGeometricTransform

This module builds a 4x4 geometric transformation matrix capable of translation, scaling, rotation, sheering and permutation.

Detailed Description

19.10.1 Default

The default transformation matrix offers no translation, scaling, rotation, shear, or permutation. The output under default settings is a 4x4 identity matrix. By default the module also only applies a single transformation option. For example, if the translate feature is applied the result will only translate, once the user switches to another option (say rotation) the output transformation matrix will return to the original identity matrix and only apply rotation. In order to apply multiple transformations, the user must first click the "composite transform" button so that the transformation remains. Once this button is pushed, the output transformation now becomes the default matrix.

19.10.2 Input Port

The module accepts a 4x4 matrix as an input transformation matrix and populates the output to reflect the input matrix. If a matrix of any other size is used as an input, no error is thrown but the matrix is not recognized - resulting in the same, default 4x4 identity matrix.

19.10.3 UI Features:

Each UI feature is selectable via radio button.

Translate

Translate allows the user to define the x, y, and z directions for translation via UI sliders.

Scale

Scaling is manipulated by logarithmic sliders at the bottom of the page. A Log Calculator (base 10) is provided at the bottom of the window for convenience.

Above the scale sliders, translation options are also given as X, Y, and Z, sliders; however, the translational shift is also determined by the log scale factor of the respective cardinal direction. For example, the x translation is augmented or diminished by the Log ScaleX slider.

Rotate

Rotation allows the user to define the angle (theta) of rotation on the bottom most slider bar. This rotation angle is applied to a portion of the rotation axes (labeled "Rotate Axis X/Y/Z"). For example, if the Rotate Axis Z is set to 1.00 and the Rotate Theta angle is set to 90. The transformation matrix will rotate an object around the Z axis by 90 degrees (centered at the origin - more on this below). If the X and Z axes are equal to each other, the transformation gives equal weight to rotation in the X and Z directions, no matter the magnitude of each slider value.

The origin can be defined by the slider bars above under the heading "Rotation Fixed Point." If X, Y, and Z are set to 0 in this section, the origin corresponds to the original origin of whatever mesh the module is applied to.

Shear

The shear stress component of the module allows the user to define a shear stress vector in the upper panel of the UI. The vector is defined using X, Y, and Z force components. The lower half of the component has allows the user to define fixed plane components with A, B, and C representing the X, Y, and Z directions. The final component D defines the offset from the origin (in the direction of the stress vector) after the stress is applied.

Permute

The permute feature allows the user to flip axes. With the FlipX/FlipY/FlipZ commands, the signs of the respective axis are reversed. With the Cycle+/Cycle- options, the axes themselves are alternated (ex. x = y, y = z, z = x). This does not actually change the coordinate system of the field that is being translated. It only alters the matrix to flip over the axis lines.

Widget

In order to use the widget feature, the visualization output port must be attached to a viewer window. Once attached a small cubic widget appears on the screen. The widget can be scaled using the Uniform Scale slider (but it does not allow for singe directional changes i.e. you cannot scale in only the X direction. . . all cardinal directions scale together). Two other features appear in the UI: 1 - Resize Separable adds rotation spheres to the widget (more on this below) and 2 - Ignore changes allows the user to place, scale, and adjust the widget as they choose without applying assigning the transformations in the output matrix. A user may use the Ignore feature to adjust the size and shape of the widget to fit their geometry before they actually start manipulating the transformation.

Once the widget is in place, the user can manipulate it in the viewer window. By holding the shift key and clicking on a portion of the widget, the user can "grab" the widget and move it. Caps Lock also works to capture the widget. If the Resize Separately box is checked in the UI, spheres appear at the center of each of the 6 faces of the cube. Additionally, the user can turn on these spheres by clicking on the widget while holding down the shift+opt key on a mac (EDIT REQUIRED TO ACCOUNT FOR WINDOWS AND LINUX CONTROLS).By so clicking, the user cycles through all of the widget options (turning them on and off with each click). Shift+clicking each part of the widget allows for the following transformations:

Gray Widget Border:

–Translation

Blue Sphere:

–Rotation

Green Cylinders:

- Scaling (if the widget has not been rotated while Ignore Changes was applied)
- Shearing (if the user has rotated the widget with the Ignore Changes applied).

19.11 CreateMatrix

This module lets the user create a small dense matrix manually.

Detailed Description

This module lets the user create a small dense matrix manually. In the GUI, one can type the contents of an arbitrarily sized matrix. First set the dimensions of the matrix, then type its contents. After setting matrix dimensions, use either the Update button or press the **enter** key to update matrix size.

19.12 CreateStandardMatrix

This module lets the user create different kinds of standard matrices of any sizes like matrices of ones, zeros, nans, identity and series matrix.

Detailed Description

This module lets the user create different kinds of standard matrices of any size(mxn). Different standard matrices include matrices of ones, zeros, nans, identity and series matrix. In the GUI, one can choose the type of matrix and size of the matrix (by defining the number of rows and columns). Also to create a series matrix the user has the flexlibility to choose the starting pointer of the matrix and the step size of the matrix.

19.13 EvaluateLinearAlgebraBinary

This module performs one of a number of selectable matrix operations using the two input matrices.

Detailed Description

One of a number of binary matrix operations can be selected from the GUI and is then applied to the two input matrices to produce a new matrix that is then sent to the output port. The operations are X, +, Normalize, Select Rows, Select Columns, or Function.

"A X B" does a matrix multiply of the two matrices. The number of rows in the first matrix must match the number of columns in the second matrix. The output matrix has the same number of columns as the first matrix and the same number of rows as the second.

"A + B" performs a matrix addition. The two input matrices must be the same size. The elements are added in a piecewise fashion. This is equivalent to selecting "Function" and using 'x+y' as the function description

"Normalize A to B" computes the min and max values of the second matrix, and then linearly transforms all of the elements in the first matrix so that they fall within the range of those min and max values.

"Select Rows" uses the values in the B input matrix as row indices to extract from the A input matrix. The B input matrix must be a Column Matrix containing valid row index values into the A matrix. For example, if the B matrix contains [3 5] and the A matrix is 100x200, then the resulting matrix will be 2x200 and contain rows 3 and 5 from the A matrix.

"Select Columns" uses the values in the B input matrix as column indices to extract from the A input matrix. The B input matrix must be a ColumnMatrix containing the valid column index values into the A matrix. For example, if the B matrix contained [3 5] and the A matrix is 100x200, then the resulting matrix will be 100x2 and contain columns 3 and 5 from the A matrix.

"Function" allows an arbitrary function of two variables to be evaluated for each number pair in the two input matrices. This requires that the two matrices are the same size. The variable representing the element from the A matrix is 'x', and the variable for the element from the B matrix is 'y'. The function is specified using SCIRun's simple function parser. There are a number of mathematical functions available for use. They are:

Operators:

- +: Add two numbers:

```
4+3 = 7
```

- -: Subtract one number from another:

```
4-3 = 1
```

- Multiply two numbers:

```
4*3 = 12
```

- Divide one number from another:

```
12/3 = 4
```

- sin: Sine of a number in **radians**:

```
sin(x)
```

- cos: Cosine of a number in **radians**:

```
cos(x)
```

- sqrt: Square root of a number:

```
sqrt(4) = 2
```

- sqr: Square of a number:

```
sqr(2) = 4
```

- ln: Natural logarithm of a number:

```
ln(x)
```

- exp: e raised to the nth power:

```
exp(ln(x)) = x
```

- log: Log base 10 of a number:

```
log(100) = 2
```

- abs: Absolute value of a number:

```
abs(-3) = 3
```

- pow: One number raised to the power of another:

```
pow(3, 2) = 9
```

- random: Return a uniform random number between 0 and 1:

```
random()
```

19.14 EvaluateLinearAlgebraUnary

Performs one of a number of selectable unary matrix operations to the input matrix.

Detailed Description

One of a number of unary matrix operations can be selected from the GUI and is then applied either to the whole matrix or to each element in the matrix, depending upon which operation is selected. The operations are Round, Ceil, Floor, Normalize, Transpose, Sort, Subtract Mean, or Function.

Round, **Ceil**, and **Floor** are used to convert each element in the input matrix into an integer value.

Normalize linearly transforms each element such that they all end up between 0 and 1.

Transpose constructs a new transpose matrix containing the same elements as the input matrix. For example, if the input matrix was of size 4x20, the output matrix will be of size 20x4.

Sort does an insertion sort on the elements of the input matrix. The elements will be sorted such that the smallest element will be at the top of the matrix and the largest element will be at the end. If the matrix is not a row or column matrix, the matrix is sorted in a row-scanline order. For example, in a 2x30 matrix, the first row would contain the lower half of the values and the second row would contain the upper half.

Subract Mean computes the mean value of the matrix, and then subtracts that value from each element in the matrix.

Function allows an arbitrary function to be evaluated for each element in the matrix. The current value is represented as the variable 'x'. For instance, the default 'x+10' function adds 10 to each element in the matrix. A function of just '10' would set each element in the matrix to be 10. The function is specified using SCIRun's simple function parser.

There are a number of mathematical functions available for use:

19.14.1 Operators

- +: Add two numbers:

```
4+3 = 7
```

- -: Subtract one number from another:

```
4-3 = 1
```

- Multiply two numbers:

```
4*3 = 12
```

- Divide one number from another:

```
12/3 = 4
```

- sin: Sine of a number in **radians**:

```
sin(x)
```

- cos: Cosine of a number in **radians**:

```
cos(x)
```

- sqrt: Square root of a number:

```
sqrt(4) = 2
```

- sqr: Square of a number:

```
sqr(2) = 4
```

- ln: Natural logarithm of a number:

```
ln(x)
```

- exp: e raised to the nth power:

```
exp(ln(x)) = x
```

- log: Log base 10 of a number:

```
log(100) = 2
```

- abs: Absolute value of a number:

```
abs(-3) = 3
```

- pow: One number raised to the power of another:

```
pow(3, 2) = 9
```

- random: Return a uniform random number between 0 and 1:

```
random()
```

19.15 ReportColumnMatrixMisfit

Compute and visualize error between two vectors.

Detailed Description

19.16 ReportComplexMatrixInfo

This module enables report field info on complex data such that complex analytics can be performed.

Detailed Description

19.17 ReportMatrixInfo

This module reports the attributes of matrices.

Detailed Description

ReportMatrixInfo is purely an informational Module. It performs no modification on input data. Upon execution it displays attributes about the input matrix in the UI.

Matrix Attributes displayed:

- Name - The Matrix name. May be blank.
- Generation - The Matrix internal object id.
- Typename - The C++ typename of the input Matrix Type.
- Rows - The number of rows in the input matrix.
- Columns - The number of columns in the input matrix.
- Elements - The size of the matrix. This is Rows x Columns for non sparse matrices. For sparse matrices this is the number of nonzero elements in the array.

19.18 ResizeMatrix

This module lets the user resize their matrix to the desired size.

Detailed Description

This module lets the user resize their matrix to the desired size. In the GUI, one can choose the size of the matrix (by defining the number of rows and columns). If the number of elements in the original matrix is less than the (no. of column)X(no. of rows) then the new matrix will be padded with zeros. If the number of elements in the original matrix are more than the (no. of column)X(no. of rows) then the matrix will be truncated. Users can also choose between the row or columns major ordering of the resizing.

19.19 SelectSubMatrix

This module chooses a submatrix based on user input. The matrix may be chosen by column and/or row ranges through the user interface or by array or matrix inputs defining desired row or column indices.

Detailed Description

There are two methods to chose a submatrix:

User Interface method: The user may select a range of rows/columns by opening the user interface window, selecting the check box associated with rows or columns and providing the desired range. Default values are -1 and does not clip the matrix.

Input Matrices method: Two additional input ports allow the user to define arrays or matrices of desired matrix indices. If the row or column matrices are nx1, the module will output a new matrix with rows/columns defined by the values in of the array. If matrices are NxM it will output a new matrix with rows/columns defined by the matrix values in a left-to-right top-to-bottom manner. If the input matrix has non-integer values, the number will be truncated (not rounded) and only the integer value will be used.

Examples:

- *Removing columns:* Columns can be removed from the front or the back of the matrix using the user interface. Alternatively, a matrix can be fed into the 'Matrix ColumnIndices' input port and will be defined by the rules in the above paragraph. Rules apply to the row input port.
- *Re-ordering rows:* Rows can be re-ordered by supplying the 'Matrix RowIndices' with a matrix that defines the order in which the user wants the rows to be ordered. Input port ordering is defined in the paragraph above. Rules apply to the column input port.
- *Cherry-picking matrix values:* By supplying matrices to both 'Matrix ColumnIndices' and 'Matrix RowIndices' values for the output matrix will be defined by the associated row and column values.

19.20 SetSubmatrix

Redefines a specified region of an input matrix to new, user-defined values.

Detailed Description

This module will replace existing values within a matrix with new values defined by the user. The module has 3 input ports. The first input port reads in the matrix the user wishes to change. The second port reads in the values that the user wishes to place inside the existing matrix. The third port is optional and reads in a 2x1 or 1x2 matrix that defines at which row and column the replacement will begin. The first number in the third port input matrix defines the initiating row. The second defines the column. If the third port is left empty the module defaults to row 0 / column 0, but these values can be altered manually within the UI.

Example 1:

Input port 1 =

```
[1 9 -13; 20 5 -6]
```

Input port 2 =

```
[a b]
```

Input port 3 =

```
[1 1]
```

Output =

```
[1 9 -13; 20 a b]
```

Example 2:

Input port 1 =

```
[1 9 -13; 20 5 -6]
```

Input port 2 =

```
[a; b]
```

Input port 3 =

```
[0 1]
```

Output =

```
[1 a -13; 20 b -6]
```

Note: The size of the input matrix cannot be altered by the size and starting points of the replacement and position matrices. In other words, if the resulting matrix would have larger dimensions than the original input matrix, the module with throw and error.

19.21 SolveLinearSystem

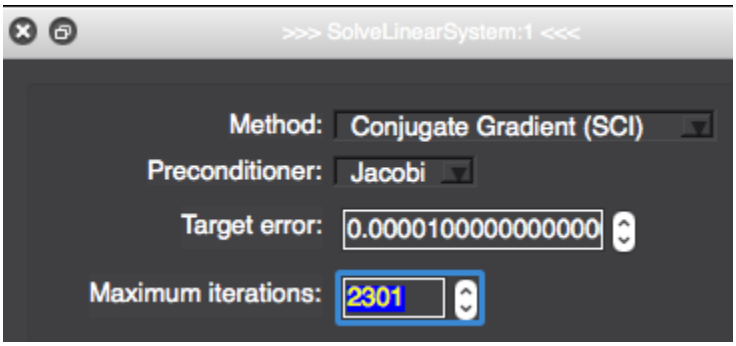
The SolveLinearSystem module is used to solve the linear system $Ax = b$, where the coefficient matrix A may be dense or sparse, b is a given right-hand-side vector, and the user wants to find the solution vector x

Detailed Description

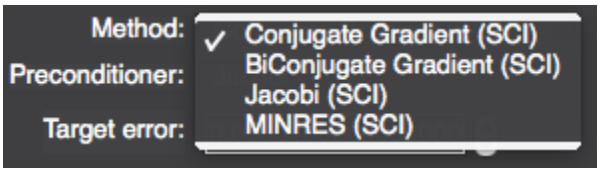
The SolveLinearSystem module takes two input matrices and returns three matrices. The first input port takes the coefficient matrix A, which may be a dense $n \times n$ matrix or a sparse $n \times n$ matrix. The second input port takes the right hand side vector b as an $n \times 1$ dense matrix. Here, the module is assuming that an $n \times n$ system is being solved.

The first of the three output ports returns the solution vector x as an $n \times 1$ dense matrix. The second output port returns the number of iterations required to reach convergence as a 1×1 dense matrix and the third output port returns the norm of the residual vector, again as a 1×1 dense matrix.

The GUI for this module is used to define the solution method for the module and monitor the convergence towards the solution.



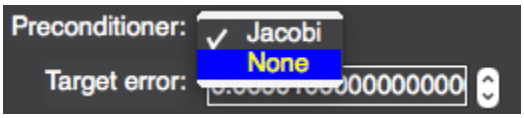
19.21.1 Methods Tab



The Methods tab allows the user to select one of four solution algorithms for numerically solving sparse systems of linear equations:

1. Conjugate gradient
2. Biconjugate gradient
3. Jacobi
4. MINRES

Each solution method comes with the option of choosing a preconditioner for the numerical solution algorithm, which is set with the Preconditioners tab:



At present, only the Jacobi preconditioner is available. However, there is the option of using no preconditioning.

19.21.2 Convergence Criteria

The next section of the GUI sets up the convergence criteria for the numerical solution.

To achieve convergence of the numerical solution, the norm of the residual must be less than the target error as given in the slider. The next slider sets the maximum number of iterations that are allowed to achieve solution. This can take a value anywhere between 0 and 20000.

The module has the ability to write solutions at regular intervals and this flag is set by clicking on the **Emit partial solutions** check-box. The frequency of writing the solutions is set with the next slider.

Finally, the option **Use the previous solution as initial guess**, which is set by clicking the check-box, allows the user to either continue a simulation run looking for better convergence, or run a second simulation for which it is expected that the second solution may be close to the first.

A useful strategy for solving simulation problems is to begin with a small number of iterations to check that the solution will converge and then use the **Use the previous solution as initial guess** option to continue the simulation. Alternatively, to stop a diverging, or at least non-converging, simulation the Target error can be dynamically changed to a residual error level already obtained by the solver.

19.21.3 Iteration Progress

The bottom section of the GUI shows how the solution is progressing.

There is an iteration counter to show how many iterations have been completed, the value of the original error (this will be 1.0 unless the **Use the previous solution as initial guess** option has been used) and the value of the current error. These quantities are summarised graphically in the convergence plot.

Further reading: [Saa03].

19.22 SolveMinNormLeastSqSystem

This module computes the minimal norm, least squared solution to a Nx3 linear system.

Detailed Description

Given four input ColumnMatrices (v0,v1,v2,b), it finds the three coefficients (w0,w1,w2) that minimize:

```
| (w0v0 + w1v1 + w2v2) - b |
```

If more than one minimum exists (the system is under-determined), the coefficients such that (w0,w1,w2) has a minimum norm is selected.

The outputs are a vector (w0,w1,w2) as a row-matrix and the ColumnMatrix (called x), which is:

```
[ w0v0 + w1v1 + w2v2 ]
```

CHAPTER

TWENTY

MISC FIELD

20.1 BuildMappingMatrix

Build a mapping matrix; a matrix that says how to project the data from one field onto the data of a second field.

Detailed Description

This module builds a mapping matrix which contains information about how to interpolate the data from the source field onto the geometry of the destination field. The resulting mapping matrix can then be used by the *ApplyMappingMatrix* module to map the values from a field similar to the source field onto the destination field.

The source field for BuildMappingMatrix and *ApplyMappingMatrix* must contain the same geometry and data locations, however they do not have to contain the same data values or value types. For instance, both a TetVolField of doubles with data at the nodes and a TetVolField of Vectors with data also at the nodes can be used as input fields for BuildMappingMatrix.

ApplyMappingMatrix can be used instead of BuildMappingMatrix and ApplyMappingMatrix if the mapping is not to be reused, as it does the same interpolation without building the intermediate mapping matrix. This should be used if the source field changes much more often than the destination field.

20.2 BuildMatrixOfSurfaceNormals

This module calculates area weighted normal vectors per node.

Detailed Description

For each node in the input surface mesh, find the attached faces, and average the face normal weighted by area for that node. Output a Nx3 DenseMatrix where N is the number of nodes in the input surface mesh. This matrix can be passed to *SwapFieldDataWithMatrixEntries* to map this data back to a vector field with the same mesh.

20.3 CalculateMeshCenter

The module computes the center of a mesh based on a predefined method, chosen by the user.

Detailed Description

The module computes the center of a mesh and outputs it as a single node. The user can chose the method with which the center is determined from the following options-

Average of Node Locations Node locations within the defined mesh are averaged. The output center will favor areas of high node density.

Average of Element Locations Element locations within the defined mesh are averaged. The output center will favor areas of high element density.

Volumetric Center (Default) Calculates the center of mass for the defined mesh, irrespective of node location, element location or Cartesian, bounding box.

Bounding Box Center Calculates the center based on Cartesian bounding box dimensions. That is, a rectangular prisim region, that completely encompasses the mesh will be generated. The mesh center will be defined as the center of the rectangular prism mentioned above.

Middle Index Node Defines the center as the central node index number. Consider a LatVol mesh of dimension 4 x 4 x 4. Such a mesh would have node index numbers 0 thru 63. The center would be defined as node number 32.

Middle Index Element Defines the center as the central element index number. Consider the same 4x4x4 LatVol mesh from above. Such a mesh would have element index numbers 0 through 26. The center would be defined as element number 13.

Note: The above Middle Index examples would have the center defined at the volumes edge (Middle Node) or at the volumes center (Middle Element). Mesh centers are determined entirely by index numbering.

20.4 GetMeshQualityField

Calculates mesh quality based on several generally accepted algorithms. Default: Scaled Jacobian

Detailed Description

Mesh quality reporting options: Jacobian/Scaled Jacobian – Jacobian and Scaled Jacobian are based on accepted mesh quality metrics as calculated by the Verdict software library. In brief, for a single node, the Jacobian matrix is defined as:

```
                    | x_1 - x_0    x_2 - x_0    x_3 - x_0 |
A_0 =               | y_1 - y_0    y_2 - y_0    y_3 - y_0 |
                    | z_1 - z_0    z_2 - z_0    z_3 - z_0 |
```

The minimal determinant of these matrices for each node of an element is known as the 'Jacobian' of the element. (Callahan et al. 2009)

Volume – Returns the volume of each element.

Scaled Inscribed/Circumscribed Ratio – Mesh quality is determined by the ratio between the radii of the largest possible inscribed sphere compared to the smallest possible circumscribed sphere of each tetrahedral element. The radius is normalized against he ratio of an equilateral tetrahedron 1:3.

20.5 ReportFieldGeometryMeasures

Build a densematrix, where each row is a particular measure of the input Field (e.g. the x-values, or the element size).

Detailed Description

ReportFieldGeometryMeasures outputs a NxM matrix filled with values associated with the input mesh measures at selected mesh locations.

First the user selects the measues location in the top half of the UI. The number of measure locations equals the number of rows (N) in the output matrix.

Then the user selects what measures are sampled at each location in the bottom half of the UI. The number of measures selected equals the number of columns (M) in the output matrix.

20.5.1 Description of Measures:

- **X position** - The X coordinate of the center of the measure location.
- **Y position** - The Z coordinate of the center of the measure location.
- **Z position** - The Z coordinate of the center of the measure location.
- **Index** - The index used by the code to reference the measure location.
- **Valence** - The # of similar measure locations that share a connection with the measure location. This varies by mesh and is not defned in all circumstances and thus equals 0 in these cases.
- **Size** - The Euclidean Volume of the measure element. Ex: length for edges or 0.0 for points.

20.6 ReportFieldInfo

This module is used to view the attributes of fields.

Detailed Description

ReportFieldInfo is purely an informational Module. It performs no modification on input data. Upon execution it displays attributes about the input field in the UI.

20.6.1 Field Attributes displayed:

- **Name** - The Field name. May be blank.
- **Generation** - The Field internal object id.
- **Typename** - The C++ typename of the input Field Type.
- **Center** - The X,Y,Z coordinates of the average center of the nodes in the Field.
- **Size** - The X,Y,Z coordinates of the grid-aligned bounding box that encloses the Field.
- **Data min,comax** - Only valid for scalar type input fields. Prints the respective min and max scalar values of the data associated with the input field.
- **Nodes** - The number of X,Y,Z points in the input field.
- **Elements** - The number of highest dimension elements in the input field. Ex. For a PointCloudField this would equal the number of nodes. For a TetVolField this would equal the number of tetrahedral cells in the input field.
- **Data at** - The location of the data values associated with the input field.

20.7 ReportNrrdInfo

This module is used to view the attributes of Nrrd data.

Detailed Description

Nrrd is both a library and a file format. The full documentation can be found at http://teem.sourceforge.net/nrrd/index.html

ReportNrrdInfo is purely an informational Module. It performs no modification on input data. Upon execution it displays attributes about the input Nrrd in the UI.

20.7.1 Field Attributes displayed:

- **Name** - The Nrrd name.
- **Type** - The data format.
- **Dimensions** - The dimensionality of the data.
- **Origin** - The position of the center of the data.
- **Spacing** - The amount of space between each point along a given axis.
- **Size** - The number of samples along a given axis.
- **Min** - The lower bound of the size in model space.
- **Max** - The upper bound of the size in model space.

There are more attributes that are not reported from this module. The other attributes are listed at http://teem.sourceforge.net/nrrd/format.html.

CHAPTER

TWENTYONE

NEW FIELD

21.1 CalculateNodeLocationFrequency

Only Point Clouds are supported at this time.

Detailed Description

21.2 ClipFieldByFunction

This module selects a subset of a field.

Detailed Description

The ClipFieldByFunction module clips out a subset of a field by preserving all of the elements in the input field for which the user specified expression evaluates to true at the specified test location. As a side effect of this algorithm is that any degenerate non-element items in the input field are discarded by the clip.

The type of the field is preserved in the clip, as well as the data values if possible. Some field types are structured and thus not clippable. This includes the LatVolField, ImageField, and Scanline field types.

The expression should be a SCIRun parser expression. The module UI has a help button that will bring up documentation for the parser which can also be found *here*. For node centered data, select clipping location from One Node, Most Nodes or All Nodes. For cell centered data, select Element Center.

The ClipFieldByFunction module takes an **input** field, and optionally a function string and/or one or more additional matrices that may be used in the function.

The ClipFieldByFunction module has two **output** ports: the clipped field and a mapping matrix.

21.3 ClipFieldByMesh

This module clips a mesh to another mesh.

Detailed Description

Clip the first input to the second mesh. Both meshes need to be volumetric meshes such as tetrahedral rather than surface meshes.

The output is the clipped mesh and a mapping matrix that allows you to apply data to the clipped mesh from the original using the module *ApplyMappingMatrix*.

21.4 ClipVolumeByIsovalue

This module clips a scalar field to a specified isovalue.

Detailed Description

The ClipVolumeByIsovalue module is used to clip a TetVol, HexVol or TriSurf field along a particular isovalue. The isovalue is specified by the entry in the GUI. The new field can contain either the values less than or greater than the isovalue, as selected by the user.

In order to compute where the cuts are to be made, the input field must contain scalar values at the nodes. If the input field contains element centered data, use modules in the ChangeFieldData category (*MapFieldDataFromSourceToDestination*, *MapFieldDataOntoNodes*, *MapFieldDataFromElemToNode*, etc.) to interpolate data from elements to nodes.

Only TetVol, HexVol and TriSurf fields (unstructured, irregular mesh types) are supported by this general clipping. Other fields should be converted into these types if they are to be clipped.

21.5 ConvertMatricesToMesh

This module constructs a mesh from raw matrix data.

Detailed Description

ConvertMatricesToMesh takes in raw position and connectivity data as matrices and uses them to construct a new mesh. The 'Mesh Positions' data should be an Nx3 matrix where N is the number of points in the mesh and each row contains data in XYZ order. The connectivity data should be an MxP matrix where M is the number of elements in the mesh and P is the number of node references per element. Thus for a TetVolMesh, P would be 4. The node references are row indices into the 'Mesh Positions' matrix and are indexed starting at zero.

The output field will contain the new mesh and no data. Modules under the ChangeFieldData category, such as *CreateFieldData* or *SetFieldData* module be used to attach data to the new field. See also modules in the ChangeMesh category, such as *SetFieldNodes*, which can be used to attach new positional data to existing meshes but cannot construct a new mesh nor change it's connectivity information.

21.6 CreateImage

This module makes an **ImageField** that fits the source field.

Detailed Description

Makes an ImageField that fits the source field. The value type of the ImageField is same as that of the input field. If there is no input field specified, then it creates a unit volume of doubles. The size parameters refer to the number of nodes in the volume, not the number of faces. Thus a 2x2x2 node lattice will only contain one facet.

The **Pad Percentage** parameter is an optional parameter that describes how much larger than the input field the resulting lattice volume should be. For example, a value of 100 would make the image field be three times as far across and contain nine times the area of a image with the default 0 padding. A value of 50 would cause the image to be twice as far across (50% bigger on each side).

No data is generated. In order to map the data mapping modules from SCIRun *MapFieldDataOntoElements* or *MapFieldDataOntoNodes* can be used if those values are needed.

21.7 CreateLatVol

This module makes a **LatVolField** that fits the source field.

Detailed Description

Make a LatVolField that fits the source field. The value type of the LatVolField is the same as that of the input field. If there is no input field specified, then create a unit volume of doubles. The size parameters refer to the number of nodes in the volume, not the number of cells. Thus a 2x2x2 node lattice will only contain one cell.

The **Pad Percentage** parameter is an optional parameter that describes how much larger than the input field the resulting lattice volume should be. For example, a value of 100 would make the lattice volume be three times as far across and contain twenty seven times the volume of a lattice with the default 0 padding. A value of 50 would cause the lattice to be twice as far across (50% bigger on each side).

No interpolation is done onto the new field. It is recommended that the *MapFieldDataOntoElements* or *MapFieldDataOntoNodes* module be used if those values are needed.

21.8 ExtractSimpleIsosurface

This moudle extracts an isopotential surface from a scalar field.

Detailed Description

The ExtractIsosurface module is used to extract one or more isopotential surfaces from a scalar field using the Marching Cubes algorithm. The isopotential surfaces are surfaces for which the scalar value would interpolate to a constant isovalue. The module can output a SCIRun field and geometry (for faster module execution, if only one is needed, deselect the output type that is not needed). The isovalues can be specified in several ways.

21.8.1 Slider

The Slider tab allows the user to select one isovalue using a slider bar or to type in one isovalue manually. The slider bar ranges from the minimum and maximum values of the input field. This can be used with great effect to interactively move the isosurface around when the **Update Auto** option is specified. Other update options are **On Release**, which updates the isosurface when the slider button is released, and **Manual**, which updates the isosurface when the module is executed.

21.8.2 Quantity

The Quantity tab allows for the selection of several regularly spaced isosurfaces (the list of isovalues is not editable). This is particularly useful for isocontouring as it gives a contour-map effect for where the isosurfaces would be located.

21.8.3 List

The List tab allows the user to type in an arbitrary spaced delimited list of isovalues to be used. In addition to hard numbers, percentages relative to the field minimum and maximum may be specified as well. For instance, 50% would specify the isovalue in the center of the min-max range, whereas 50.0 would isosurface at a value of 50.0.

21.8.4 Matrix

Isovalues can be passed into the field in the **Optional Isovalues** port, which expects a Nx1 matrix. The Matrix tab must be selected and the module executed to populate the list of isovalues from the **Optional Isovalues** input matrix. A surface will be created for each value in that matrix. The *ExtractIsosurface-probe.srn* example network demonstrates this using the *GenerateSinglePointProbeFromField* and *SwapFieldDataWithMatrixEntries* modules to interactively select the isosurface with a probe using this port.

The **Build Output Field** option determines whether or not a surface field is created. The **Build Output Geometry** option determines whether or not geometry data is created. Transparency can be enabled for geometry output by selecting the **Enable Transparency** option. If there is no **ColorMap** present then the default color is used for the output geometry.

This module can work on scalar fields with the following mesh type: LatVol, StructHexVol, HexVol, TetVol, Prism, Image, StructQuadSurf, QuadSurf, TriSurf, Scanline, StructCurve, and Curve. Isosurfaces are generated for volume meshes, isocontours are generated for surface meshes, and isopoints are generated for edge meshes.

If the scalar fields contain cell-centered data, the surfaces are created along the appropriate cell faces. In the case where the isovalue exactly equals the scalar data only the face between the cell and its neighbor with a great scalar value will be extracted.

21.8.5 Output

There are four output ports:

1. The first output port contains the field
2. The second output port is the geometry (useful for large data sets if the extracted surface will only be viewed)
3. The third output port contains a mapping matrix to the nodes used to build the isosurface
4. The fourth output port contains a mapping matrix to the cells used to build the isosurface.

21.9 FairMesh

This module smooths surface meshes without shrinking them.

Detailed Description

Based on the surface smoothing algorithm published by Taubin in 1995 [Tau95].

User Interface:

- Weighting methods: Equal and curvature normals (default: equal)
- Iterations: Number of times the surface is put through the smoothing algorithm (default: 50)
- Spatial cut off frequency: Similar to low pass filter setting (default: 0.1)
- Relaxation Parameter: negative scale factor – the shrinking term (default: 0.6307)
- Note: an unshrinking term is produced by: Spatial cutoff = 1/Relaxation parameter + 1/Dilation parameter

21.10 GeneratePointSamplesFromField

This modules allows you to set seed points in a given field and interactively change their location.

Detailed Description

The **input** is a field containing a mesh.

The **Geometry** output connects directly to the ViewScene module and produces points that can be interactively moved by holding shift while dragging them. This will update the points output to the Field port.

The **Field** output the points as a point cloud field.

21.11 GeneratePointSamplesFromFieldOrWidget

This module generates samples from any type of input field and outputs the samples as a **PointCloudField** field.

Detailed Description

GeneratePointSamplesFromFieldOrWidget generates samples from any type of input field and outputs the samples as a PointCloudField field. The samples can be generated randomly or user selected via a 3D widget. Use the **shift key/left mouse button** combination to manipulate the size (resize cylinders), orientation (spheres on the widgets are rotation points) and position of the widgets. Use the same key/mouse combination to move the slider on the rake widget. Use the **shift key/right mouse button** combination to bring up a dialog box to change widget scale.

The random sampling can be weighted by importance (determined by the data value of the sample) or left unweighted. The random sampling can also be limited to the selection of node points, rather than points from the interior of the fields elements, by clamping the samples to nodes.

The **uniform** distributions are a uniformly random function over the spacial extents of the field (i.e. the elements are weighted by their volume or area), whereas the **scattered** distributions are a uniformly random function over the elements only (i.e. all elements have the same weight). The **importance weighted** distributions multiply the existing weight of a sample by the interpolated data value associated with it, while the **not weighted** distributions leave the existing weight of a sample unchanged.

21.12 GenerateSinglePointProbeFromField

This module generates SinglePointProbeFromField values in a field.

Detailed Description

A GUI Point widget is created and can be moved around in the scene. Use the **shift key/left mouse button** combination to move the widget.

The GenerateSinglePointProbeFromField Point port contains a **PointCloudField** field with one point in it at the location of the probe. The value at that point is interpolated from the input field. The input field is also used to determine the initial position and relative size of the probe widget. If there is no input field, a default zero valued probe is returned.

The **Element Index matrix** contains the Index of the item with data that is nearest the cell. If an input field is not available, then there will not be any matrix output.

21.13 GetCentroidsFromMesh

Computes a point cloud field containing all of the elements for a field.

Detailed Description

The Centroids module computes a point cloud field containing all the element /node/face/edge/cell/delement centers for a field. For instance, if the input field is a TetVolField then the output field would contain the center of each tetrahedra. An input point cloud field would return the same data locations.

The output field is always a point cloud field of doubles, and contains no data. If data is needed at those points then a mapping module (*MapFieldData* module) could be used to recover them.

21.14 GetDomainBoundary

This module will extract the inner and outer boundaries from a mesh.

Detailed Description

An **inner boundary** is defined as the boundary between different regions with a different value on the elements.

The **outer boundary** is the boundary that surround the mesh.

In the GUI one can selectively set which boundaries are extracted from the mesh. One can set the range of values on the segmented field for which one wants to extract the boundary separating the different compartments. Within this range of selected domains one can select only the inner boundaries to this selected domain or one can get the boundary surrounding the selected domains.

This module is intended to extract the boundaries in a segmented field. The module is templated and should work on any mesh type as long as the data is assigned to the elements.

21.15 GetFieldBoundary

This module extracts a boundary surface from a volume field.

Detailed Description

The GetFieldBoundary module extracts the bounding surface of the incoming field, making it into a new field that it outputs through the BoundaryField port. This module does not have a GUI. It has one input port, Input, and two output ports, **Boundary**, and **Mapping**. The GetFieldBoundary module builds an appropriate mesh from the incoming field. GetFieldBoundary will build a QuadSurfMesh, TriSurfMesh, or CurveMesh as appropriate for the boundary type of the input field. The Boundary output port will contain the new boundary mesh with no data interpolated onto it. If the data from the input field is desired on the output field, it is recommended to use the **Mapping** output, running the results through *ApplyMappingMatrix* in order to draw values from the input field.

Any interior face should be shared by two cells. The boundary surface is calculated as the module checks each face that is shared by two or more cells. The faces that exist in one cell only are boundary faces. This module can be used as a debugging tool because errors or holes in the mesh will show up as boundary faces.

21.16 GetSliceFromStructuredFieldByIndices

This module reduces the dimension of a topologically regular field by 1 dimension.

Detailed Description

The GetSliceFromStructuredFieldByIndices **input** port is a field that has regular topology. The field will be reduced by 1 dimension and piped to the output port.

The user must supply two input values either via the module GUI, or from an optional input matrix: the axis the field is reduced along, and location along the selected axis. If supplying an matrix of clipping locations, the matrix must either be 1x1 or 3x3 and have data entries that follow this pattern:

- If 3x3 matrix, row index is the axis: row 0 = axis i, row 1 = axis j, row 2 = axis k (only select one at a time)
- Column 0 is the selected axis to slice
- Column 1 is the slice index (location along the selected axis where the field will be sliced)
- Column 2 is the data dimensions

A 3x3 matrix containing selected axis, slice index and data dimensions following the above pattern is also output.

21.17 InterfaceWithCleaver

Takes a field entry and converts it into a Cleaver2 mesh using the parameters listed in the module UI.

Detailed Description

21.18 InterfaceWithTetGen

This module is a port to open and run Tetgen, a delauney tetrahedralization software package, within SCIRun.

Detailed Description

InterfaceWithTetGen is a module that will make a tetrahedral mesh from a trisurf mesh or a point cloud. This is the easiest way to turn a surface mesh into a volume in SCIRun. Using delauney tetrahedralization, tetgen will find the tetrahedral mesh that will connect the input points with the highest quality elements, i.e., the element face area are as equal as possible. For more information about tetgen an its capabilities, please refer to the TetGen website.

In order for tetgen to run effectively, the flags must be properly set. If using a surface input, the mesh must be a valid trisurf mesh that is completely inclose and without overlapping/crossing elements. If using only a point cloud input, the first option (Tetrahedralize a piecewise linear complex (PLC)) must be disabled. In the case of a point cloud only input, the output will be a convex hull of the points. If this module is taking too long (several minutes) and you are not sure why, try getting rid of any quality or size constraints (should finish quickly, as fast as a few seconds depending on the mesh) and then gradually reintroducing them. The few the options enabled, the faster and more likely to solve the tetrahedralization.

There are four inputs:

1. **Main (Required):** The main field to be processed by tetgen. The outer surface or Tetrahedral volume field to refine.
2. **Points (Optional):** If there is a PointCloud attached to this input, the point in it will be included in the output tetvol. Data on this field is meaningless.

3. **Region Attribs (Optional):** If a PointCloud is attached the points will be considered points inside the different regions. The data will be used as the volume constraint for that region. For the volume constraint to be respected you must pass the 'a' command line switch.
4. **Regions (optional):** This is a dynamic input, each input should be a surface field, that defines a region inside the Main Input.

The module outputs a **tetrahedral mesh**.

21.19 JoinFields

This module glues any number of input fields into one output field.

Detailed Description

JoinFields takes in an arbitrary number of input fields and gathers them all up into one output field. If the input fields all have the same editable mesh type then the output field will also be of the same type. If the meshes are not editable (LatVol, Image and Scanline) the mesh type is not perserved because they cannot be arbitrarily glued together while maintaining their mesh type. Instead, appropriate equivalent mesh types will be produced (HexVol, QuadSurf).

Options available through the UI include:

1. Force PointCloudField as output
2. Merge duplicate nodes (default)
3. Merge duplicate elements
4. Only merge nodes with same value
5. Merge mesh only, do not assign values

The user may also set a tolerance defining the distance between nodes and/or elements to be merged.

21.20 SplitFieldByConnectedRegion

This module splits a domain into separate fields as defined by the input field's connectivity.

Detailed Description

This module divides a single input field into as many as 8 output regions. Separate regions may also be collected and bundled. Connected regions are not defined by label masks. They are defined, rather, by node and/or element connectivity. For example, a continuous, tetrahedral mesh may use label masks to define domains, but this module will produce only one output field given that all tets within the mesh are connected.

A common use for this module, then, is to separate distinct field surfaces, to split away small mesh islands that are not connected to the main mesh, or to break up a fractionated mesh into its individual regions.

Domains can be sorted in two ways: **Sort Domains by Size** or **Sort Ascending**. Domain size refers to the overall surface area (if domains are surfaces) or volume (if domains are 3-dimensional). The ascending sort refers to the value assigned to each domain in the original field.

21.21 SplitFieldByDomain

This module splits a domain with predefined domains (i.e. label masks) into separate fields.

Detailed Description

This module accepts a single input field with predefined domains and splits each domain into separate fields. New fields are output as individual fields, or as a bundle of distinct fields. Up to 8 distinct possible output fields are available per module. If additional splits are needed, the module can be repeated as often as necessary on the final output field port. Domains can be sorted in two ways: **Sort Domains by Size** or **Sort Ascending**.

Domain size refers to the overall surface area (if domains are surfaces) or volume (if domains are 3-dimensional).

The ascending sort refers to the label mask that defines each domain in the original field.

The resulting output will be ordered relative to the values used to define the input field's domains.

CHAPTER

TWENTYTWO

PYTHON

22.1 InterfaceWithPython

This module allows you to take an input (String, Matrix, or Field) and perform Python-based algorithms on the input code. The module UI allows you to name each of the inputs and outputs which can then be referenced in the pasted code block.

Detailed Description

A detailed description of this module is not available at this time. For assistance please contact scirun-users@sci.utah.edu.

22.2 PythonObjectForwarder

For advanced use. Most features integrated into interface with Python.

Detailed Description

CHAPTER

TWENTYTHREE

RENDER

23.1 ViewScene

This module displays interactive graphical output to the computer screen. Use the ViewScene to see a geometry, or spatial data. The ViewScene provides access to many simulation parameters and controls, thus, indirectly initiates new iterations of the simulation steps important to computational steering.

Detailed Description

Autoview restores the display back to a default condition. This is very useful when objects disappear from the view window due to a combination of settings.

Set Home View captures the setting of the current view so you can return to it later by clicking the Go home button. Go home restores the current home view.

Views lists a number of standard viewing angles and orientations. The view directions align with the Cartesian axes of the objects. The Up vector choice sets the orientation of the objects when viewed along the selected axis.

From the ViewScene window, the left corner of the control panel contains performance indicators that document the current rendering speed for the display. More advanced graphics performance results in a higher drawing rate.

The Viewer can be cloned with the **NewWindow** button in the menu. When a new window is created it is locked to the original window, meaning that rotations in one are copied to the other window. This mode is called the 'View Locking Mode' and is indicated with a little 'L' in the lower right corner. To independently rotate the viewers, release the lock by pressing 'L' in the window that needs to be unlocked. If one presses 'L' again the window will return to locking mode in which all the mouse movements are copied to all other windows that have locking mode switched on.

To improve navigation in 3D data and help facilitate the placement of widgets, it is suggested to use multiple windows that view the object from different angles. By pressing 1-8 in the window one can quickly move through different views, that look at the object from different angles.

A small plus sign (**+**) appears in the lower right corner of the ViewScene window. Clicking on the plus sign reveals the extended control panel.

23.1.1 Hotkeys

The ViewScene module also supports a series of hotkeys:

- **1-8:** Standard views that align with the cartesian axis.
- **CTRL 1-9:** Import view from Viewer Window 1-9.
- **0:** Restore display back to default view (Autoview).
- **CTRL H:** Capture current view and store it (Set Home View).
- **H:** Go to captured view (Go Home).

- **A:** Switch Axes ON/OFF.
- **B:** Switch Bounding box mode ON/OFF.
- **C:** Switch Clipping ON/OFF.
- **D:** Switch Fog ON/OFF.
- **F:** Switch Flat shading ON/OFF.
- **I:** Display latest information on hotkeys.
- **K:** Switch lighting ON/OFF.
- **L:** Switch view locking ON/OFF.
- **O:** Switch orientation icon ON/OFF.
- **P:** Switch orthographic projection ON/OFF.
- **U:** Switch backculling ON/OFF.
- **W:** Switch wireframe ON/OFF.

23.1.2 Tools

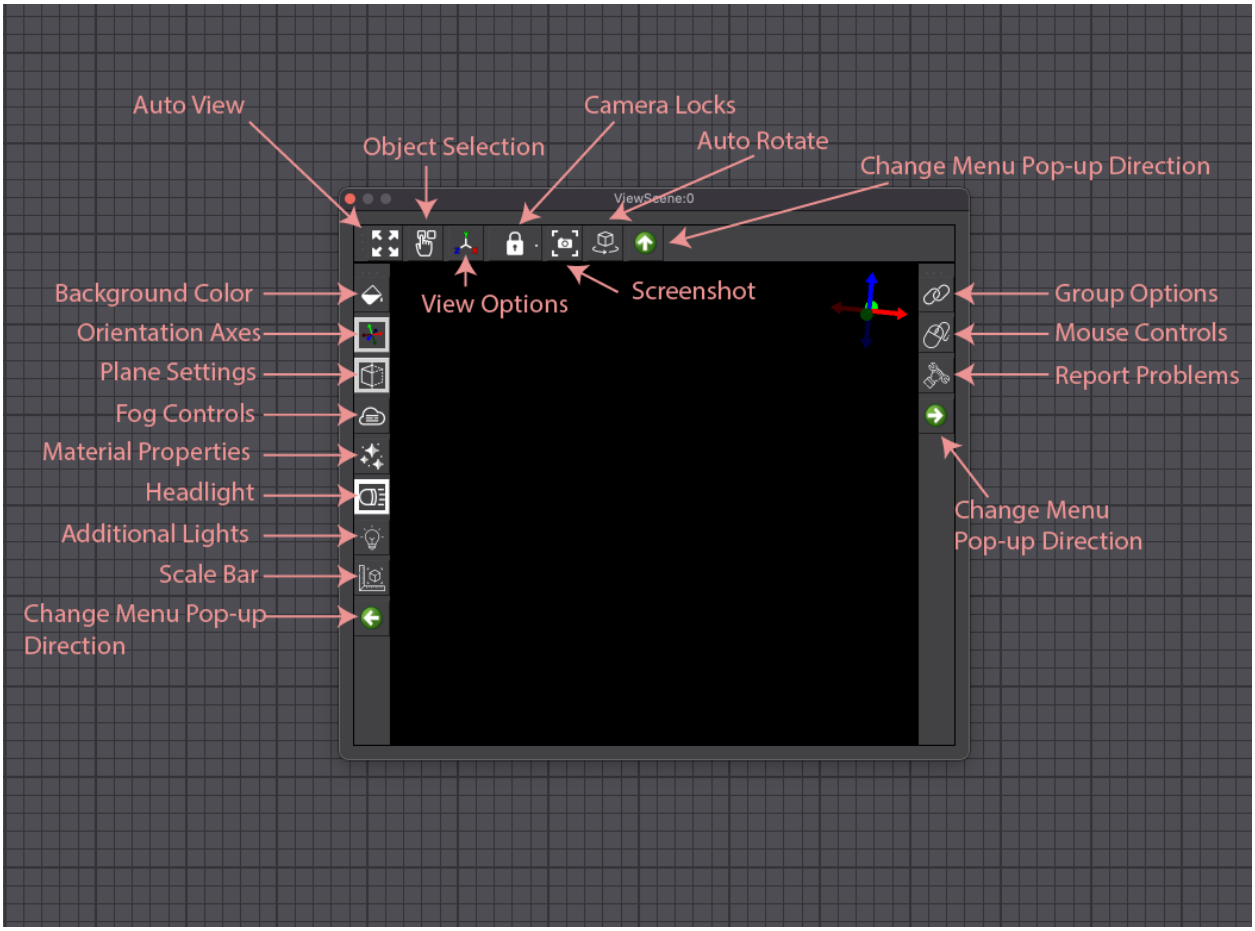


Fig. 23.1: ViewScene window controls

The ViewScene window contains three toolbars located to the top, right and left of the window (see Fig. 3.11). The tools have the following functions:

Auto View

restores the display back to a default condition. This is very useful when objects disappear from the view window due to a combination of settings

Object Selection

View Options

Camera Locks

Screenshot

Auto Rotate

Background Color

Orientation Axes

Plane Settings

Fog Controls

Material Properties

Headlight

Additional Lights

Scale Bar

Groups

Mouse Controls

Report Problems

CHAPTER

TWENTYFOUR

STRING

24.1 CreateString

This module can be used to create a string.

Detailed Description

This module allows the user to create a string by typing the string into an edit box in the GUI.

24.2 GetNetworkFileName

This module retrieves the name of the current network.

Detailed Description

GetNetworkFileName retrieves and sends the name of the current network to a String port. If a network has not been loaded, the module generates a default.

24.3 JoinStrings

This module merges multiple strings into one string.

Detailed Description

This module merges multiple strings into one string. The strings are read from the left most port to the right most port and are put into one string. (The string of the left most input port is the first string in the merged data and the one on the right is the last one).

24.4 NetworkNotes

A description of this module is not available at this time. For assistance please contact:

scirun-users@sci.utah.edu.

Detailed Description

24.5 PrintDatatype

Detailed Description

A detailed description of this module is not available at this time. For assistance please contact:

scirun-users@sci.utah.edu.

24.6 PrintMatrixIntoString

This module does a sprintf with input matrices into a new string.

Detailed Description

This module module takes in a formatted string (a string with %01d, %4.5f, %g etc) and takes the numbers of the input matrix and puts these formatted numbers into the string. The input matrix can be a scalar but can as well be a full dense matrix. The module takes each number out of the matrix and puts it in the formatted number string. For example in order to plot the contents of a 1x3 vector one may choose a formatstring of '%f %f %f\n', which will put all three value in the matrix in the formatted string. If there are more numbers in the matrix than in the format string then the current format string will be reused again and the format string is filled out from the first entry again. Hence, to get the contents of a 4x4 matrix in a comma seperated list one can choose a format string of '%5.5f, c'. This will apply this format string to each number in the matrix. In case multiple matrices are supplied, first all the values of the first matrix will be used, then all the numbers of the second matrix, etc. When there are more positions in the matrix where one can fill out a number then the rest will be padded with the value 0.0.

The format string to be used can be either entered in the GUI or on the first input port. If a string is found at the first input port, this one is automatically inserted in the widget on the GUI and is the one to be used.

Note: the format string may contain '%s', which will remain in the format string and is not touched by this module. To insert strings use *PrintStringIntoString* instead.

24.7 PrintStringIntoString

This module prints a string into a formatted string.

Detailed Description

This module performs a sprintf operation with strings, whereever a '%s' in the format string is found the input string will be pasted. This module takes in a new string for each '%s' encountered in the format string. If not enough string are supplied an empty string is used.

Note: This module leaves any numeric formatting untouched. Hence statements like '%d' or '%f' remain in the string. To insert numbers, use the *PrintMatrixIntoString* module.

24.8 ReportStringInfo

This module can be used to display the contents of a string.

Detailed Description

This module shows the contents of a string.

24.9 SplitFileName

This module splits a filename in: pathname, filename (base), and extension.

Detailed Description

This module splits the complete filename into its three base components: pathname, filename, and extension. These three string are in the first, second, and third ports respectively. To get the full filename back, just merge the three resulting strings together. The forth port contains the filename and extension (i.e. removes the base path).

CHAPTER

TWENTYFIVE

VISUALIZATION

25.1 CreateStandardColorMap

This module generates fixed Colormaps for visualization tools.

Detailed Description

This module is used to create some "standard" non-editable colormaps in Dataflow/Uintah. Non-editable simply means that the colors cannot be interactively manipulated. The module does, allow for the the resolution of the colormaps to be changed. This class sets up the data structures for Colormaps and creates a module from which the user can choose from several popular colormaps. By clicking in the Color band the user manipulate the **transparency** of the color. This is useful, for example, when volume rendering, though many visualization tools ignore the transparency data.

Most of the important work for this module is performed in the CreateStandardColorMaps.tcl file. There you can easily add new colormaps by making the obvious changes to the buildColorMaps function and the UI function (where the make_labeled_radio buttons are created). The C++ code merely queries the tcl code and fills the Colormap.

25.2 GenerateStreamLines

This module visualizes vector fields by generating curves that interpolate the vector of vectors in a Field.

Detailed Description

The **Vector Field** input port is the vector field on which the path intergration of the seed points will be performed. This may be a vector field with any volume geometry type. Surfaces and lower order fields do not currently work due to numerical error.

The second input port, **Seeds**, contains the initial values for which the vector will be computed. This may be a field of any type, as just the node positions are used. Seeds which are not contained within the vector field are just discarded. The module requires use of both field inputs in order to execute.

The GenerateStreamLines module has one output, **Streamlines**, a CurveMesh, representing a collection of stream lines computed for the Vector field given the set of initial seed points.

This module uses one of several similar numeric integration methods. They are taken directly from Numerical Analysis by David Kincaid and Ward Cheney [KC96]. Adams-Bashforth is a Multi-Step method that offers the fastest performance by reusing existing samples when possible. It presents some interesting artifacts on discontinuous models such that it is easy to visualize where those discontinuities take place. Heun is a second order Runge-Kutta solver, and is sufficient for most smoothly varying fields, such as linearly interpolated gradient fields over volume elements. Classic 4th Order Runge-Kutta is presented for reference and provides excellent results in general. Adaptive Runge-Kutta-Fehlberg is a 5/6 order solver which provides the best overall results at the cost of more computation time. It is also the only adaptive method, meaning the step size of the algorithm is adjusted dynamically as the stream moves through the vector field.

All of the computation methods are similar in that they take a step, (based upon the Step Size indicated) examine the derivative at that point, and then procede in that new direction. Adaptive Runge-Kutta-Fehlberg also determines if a stepsize correction is necessary. This determination is based upon the Error Tolerance value.

The GenerateStreamLines GUI includes text fields for **Error Tolerance**, **Step Size**, and **Maximum Steps**. The Error Tolerance text field represents the margin of error when doing an adaptive calculation of the stream lines. Maximum Steps represents the maximum number of iterations.

The user can specify whether streamlines are computed in negative and/or, positive directions by changing the Direction radio buttons. The **Color Style** options are based on either the seed number, integration step, distance from the seed, or the total distance, which will affect how values are attached to the output field. By selecting seed number, the output field will range from zero to the number of seed points so that every stream line has exactly one value. The integration step option changes the value of output field such that they start at zero where the seed point is at and go up by one for each integration step away in either direction. The distance from seed is similar to the integration step but based on the cord length distance from the seed. Whereas the total length is based on the total cord length from the seed point.

The **Filter Colinear Points** check box is a postprocess on the streams. It passes over the streams and removes all points that are too close together, as well as points that are colinear. It is useful for rendering the streamlines as it greatly simplifies them while retaining their visual appearance.

25.3 RescaleColorMap

This module allows the user to manually or automatically set the range of scalar values that map to the ColorMap. RescaleColorMap is an example of a module that has dynamic ports, because each time the user connects a Field module to the RescaleColorMap Field input port, another Field input port appears.

Detailed Description

In the GUI, the **Auto Scale** check box uses the minimum and maximum values found in the ScalarField and maps the ColorMap to that range. The **Fixed Scale** check box allows the user to manually select for the range of scalar values to which the colors map by adjusting the minimum and maximum values in the appropriate text fields. After changing the range to which the colors map, a new ColorMap passes downstream to the connecting module.

25.4 ShowAndEditDipoles

This module visualizes point clouds with vector data. It allows you to move, rescale, and rotate vectors.

25.4.1 Ports

This module has 1 input port and 2 output ports. The input port must be a point cloud mesh with vectors on all of the points. The geometry output port visualizes vectors as widgets, which can be edited through the *ViewScene* module. The field output sends the edited data downstream.

25.4.2 Editing

Each vector is visualized as 4 separate widgets. To interact with widgets, hold the SHIFT key and click and drag the widget. Moving different widgets will cause different transformations on that vector:

Part of Vector	Transformation
Sphere	Movement
Cylinder	Movement
Cone	Rotation
Disk	Resize

25.4.3 Scaling

The Visualization Scaling Factor multiplies the scaling of all the dipoles, however it does not affect the field output.

The 3 scaling options rescale the field output.

Scaling Option	Transformation
Original	Uses the scaling values given from the input port
Normalize Vector Data	Normalizes all vector
Normalize by Largest Vector	The largest vector is normalized and the rest are rescaled in proportion of the largest

Note: Changing scaling options reset to the original scales from the input field. However, you can resize the widget after changing scaling options.

25.4.4 Other Options

Option Name	Description
Show Last As Vector	Shows the last vector in the field as a point
Show Lines	Displays lines connecting between the bases of every vector in the field
Move Dipole Positions Together	When one dipole is moved, every other dipole moves the same direction and distance

25.5 ShowColorMap

This module displays a ColorMap with index values.

Detailed Description

ShowColorMap creates a geometry overlay containing the input colormap and numerical values for its range.

25.6 ShowField

This module visualizes the geometry that makes up a **Mesh** inside a Field. When possible and selected, the field takes its color from the data values that permeate the field.

Detailed Description

The field in the first input port holds the mesh that is to be visualized. By default is will be displayed using the the default colow, which is editable from the UI. If there is a color map attached to the second input port, and there is valid data in field, then the data can be used as an index into the color map, and the mesh is rendered with appropriate colors. In addition, if there is valid data is field the data itself can be converted into a color. Scalar data creates a gray scale mapping, vector data (normalized) creates RGB colors, and the principle Eigen Vector (normalized) of tensor data also creates RBG colors.

Nodes can be rendered as points (default) or as spheres. **Edges** can be rendered as lines (default) or cylinders. If the nodes and edges are rendered as spheres and cylinders respectively they may be sized. **Faces** and **Text** can be rendered or not.

25.7 ShowFieldGlyphs

This module visualizes the data that makes up a Field. When possible and selected, the glyph takes its color from the data values that permeate the field.

25.7.1 Ports

This module has 6 input ports, 3 field inputs and 3 color map inputs. The only required port is the first field input port because that holds the data that is visualized. By default, ports will be set to primary input.

Secondary and Tertiary Ports
The secondary and tertiary ports can be used to scale vector parameters like width. **Ex:** When rendering vectors as a cone the orientation and length of the cone are determined by the primary field, but the secondary field can be used to change the radius of the cone.

They can also be used to assign colors based on data besides the primary field. **Ex:** If you want to render scalars in RGB, you can give a secondary input of a vector or tensor and set the coloring to RGB Conversion through the secondary input.

25.7.2 Color

By default, glyphs is displayed using the default color, which is editable from the GUI.

Color Map
If there is a color map and field attached to the selected input port, then the data in the selected field input is used as an index into the corresponding color map, and the glyph is rendered with the color at that point.

Type	Description
Scalar	Uses scalar value as index
Vector	Uses vector's length as index
Tensor	Uses the vector magnitude of the 3 eigenvalues as the index

RGB Color Conversion
If there is a field attached to the selected input port, then the data in the selected field is converted into a color. The RGB color components are generated by the absolute value of vector x, y, z components:

```
(R = |x|, G = |y|, B = |z|)
```

Type	Description
Scalar	Creates gray-scale mapping
Vector	Normalized vector creates RGB colors
Tensor	Principle eigenvector(normalized) creates RGB colors

25.8 ShowMeshBoundingBox

The ShowMeshBoundingBox Module renders a semented red-green-blue 3D grid around an arbitrary field

Detailed Description

ShowMeshBoundingBox queries the generic mesh interface to get the BBox of the Field, and then renders a "cage" of that bounding box. The user specifies nx/ny/nz (number of "bars" in each direction) via text entry in UI. The cage is rendered with red/green/blue lines corresponding to x/y/z.

25.9 ShowString

This module puts the contents of a string in the viewer window.

Detailed Description

This module is able to display multiple lines of information into the viewer window. The module takes a string as input and displays the contents on top of the viewer window. If there are 'newlines' in the string multiple lines are displayed (up to ten lines). In the GUI the font size and color can be altered as well as the location of where to plot the text. This module is intended to provide the user with a means to put any arbitrary information about the data being shown inside the figure.

CHAPTER

TWENTYSIX

OTHER LINKS

- CIBC
- Other CIBC Softwares
- SCIRun Main

CHAPTER

TWENTYSEVEN

BIBLIOGRAPHY

CHAPTER

TWENTYEIGHT

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[BRS77] Roger C. Barr, Maynard Ramsey, and Madison S. Spach. Relating epicardial to body surface potential distributions by means of transfer coefficients based on geometry measurements. *IEEE Transactions on Biomedical Engineering*, BME-24(1):1–11, 1977. doi:10.1109/TBME.1977.326201.

[dM92] J.C. de Munck. A linear discretization of the volume conductor boundary integral equation using analytically integrated elements (electrophysiology application). *IEEE Transactions on Biomedical Engineering*, 39(9):986–990, 1992. doi:10.1109/10.256433.

[KC96] David Kincaid and Ward Cheney. *Numerical Analysis: Mathematics of Scientific Computing (2nd Ed)*. Brooks/Cole Publishing Co., USA, 1996. ISBN 0534338925.

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, USA, 2nd edition, 2003. ISBN 0898715342. URL: http://www-users.cs.umn.edu/~saad/books.html.

[SPM86] Peggy C. Stanley, Theo C. Pilkington, and Mary N. Morrow. The effects of thoracic inhomogeneities on the relationship between epicardial and torso potentials. *IEEE Transactions on Biomedical Engineering*, BME-33(3):273–284, 1986. doi:10.1109/TBME.1986.325711.

[Tau95] Gabriel Taubin. A signal processing approach to fair surface design. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, 351–358. New York, NY, USA, 1995. Association for Computing Machinery. URL: https://doi.org/10.1145/218380.218473, doi:10.1145/218380.218473.